

Android 软件开发教程

张雪梅 高凯 编著

清华大学出版社

深入浅出系列规划教材

深入浅出 Android 软件开发教程

张雪梅　高　凯　编著

清 华 大 学 出 版 社
北　　京

内 容 简 介

本书是面向 Android 初学者的教程，介绍设计开发 Android 应用程序的基础理论和实践方法，讲解 Android 系统的体系结构、Java 语言与面向对象编程基础、XML 基础、开发环境搭建、Android 应用程序的调试和发布方法、用户界面设计、组件间的通信与广播、后台服务、数据的存储和访问、图片和音视频的处理、Web 应用程序的设计等内容。本书理论与实践相结合，内容详尽，配有丰富的示例程序，讲解深入浅出，可以使读者在较短的时间内理解 Android 系统框架及其应用的开发过程，掌握 Android 应用程序的设计方法。

本书提供所有程序的源代码和电子课件。本书可作为普通高等院校及各类培训学校 Android 软件开发课程的教材，也可作为 Android 程序设计爱好者的自学用书。

图书在版编目（CIP）数据

深入浅出 Android 软件开发教程/张雪梅，高凯编著. --北京：清华大学出版社，2015
深入浅出系列规划教材
ISBN 978-7-302-40055-4

Ⅰ. ①深…　Ⅱ. ①张…　②高…　Ⅲ. ①移动终端—应用程序—程序设计—教材　Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2015)第 089365 号

责任编辑：白立军
封面设计：傅瑞学
责任校对：焦丽丽
责任印制：王静怡

出版发行：清华大学出版社
网　址：http://www.tup.com.cn，http://www.wqbook.com
地　址：北京清华大学学研大厦 A 座　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京国马印刷厂
经　销：全国新华书店
开　本：185mm×260mm　　**印　张**：21　　**字　数**：483 千字
版　次：2015 年 5 月第 1 版　　**印　次**：2015 年 5 月第1 次印刷
印　数：1～2000
定　价：39.00 元

产品编号：057447-01



丛书序

为什么开发深入浅出系列丛书？

目的是从读者角度写书，开发出高质量的、适合阅读的图书。

“不积跬步，无以至千里；不积小流，无以成江海。”知识的学习是一个逐渐积累的过程，只有坚持系统地学习知识，深入浅出，坚持不懈，持之以恒，才能把一类技术学习好。坚持的动力源于所学内容的趣味性和讲法的新颖性。

计算机课程的学习也有一条隐含的主线，那就是“提出问题→分析问题→建立数学模型→建立计算模型→通过各种平台和工具得到最终正确的结果”，培养计算机专业学生的核心能力是“面向问题求解的能力”。由于目前大学计算机本科生培养计划的特点，以及受教学计划和课程设置的原因，计算机科学与技术专业的本科生很难精通掌握一门程序设计语言或者相关课程。各门课程设置比较孤立，培养的学生综合运用各方面的知识能力方面有欠缺。传统的教学模式以传授知识为主要目的，能力培养没有得到充分的重视。很多教材受教学模式的影响，在编写过程中，偏重概念讲解比较多，而忽略了能力培养。为了突出内容的案例性、解惑性、可读性、自学性，本套书已尽力在以下方面做好工作。

1. 案例性

所举案例突出与本课程的关系，并且能恰当反映当前知识点。例如，在计算机专业中，很多高校都开设了高等数学、线性代数、概率论，不言而喻，这些课程对于计算机专业的学生来说是非常重要的，但就目前对不少高校而言，这些课程都是由数学系的老师讲授，教材也是由数学系的老师编写，由于学科背景不同和看待问题的角度不同，在这些教材中基本都是纯数学方面的案例，作为计算机系的学生来说，学习这样的教材缺少源动力并且比较乏味，究其原因，很多学生不清楚这些课程与计算机专业的关系是什么。基于此，在编写这方面的教材时，可以把计算机上的案例加入其中。例如，可以把计算机图形学中的三维空间物体图像在屏幕上的伸缩变换、平移变换和旋转变换在矩阵运算中进行举例；可以把双机热备份的案例融入到马尔科夫链的讲解；把密码学的案例融入到大数分解中等。

2. 解惑性

很多教材中的知识讲解注重定义的介绍，而忽略因果性、解释性介绍，往往造成知其然而不知其所以然。下面列举两个例子。

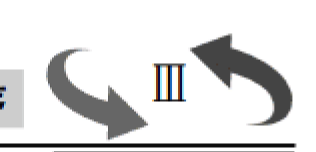
(1) 读者可能对 OSI 参考模型与 TCP/IP 参考模型的概念产生混淆，因为两种模型之

间有很多相似之处。其实，OSI 参考模型是在其协议开发之前设计出来的，也就是说，它不是针对某个协议族设计的，因而更具有通用性。而 TCP/IP 模型是在 TCP/IP 协议栈出现后出现的，也就是说，TCP/IP 模型是针对 TCP/IP 协议栈的，并且与 TCP/IP 协议栈非常吻合。但是必须注意，TCP/IP 模型描述其他协议栈并不合适，因为它具有很强的针对性。说到这里读者可能更迷惑了，既然 OSI 参考模型没有在数据通信中占有主导地位，那为什么还花费这么大的篇幅来描述它呢？其实，虽然 OSI 参考模型在协议实现方面存在很多不足，但是，OSI 参考模型在计算机网络的发展过程中起到了非常重要的作用，并且，它对未来计算机网络的标准化、规范化的发展有很重要的指导意义。

（2）再例如，在介绍原码、反码和补码时，往往只给出其定义和举例表示，而对最后为什么在计算机中采取补码表示数值？浮点数在计算机中是如何表示的？字节类型、短整型、整型、长整型、浮点数的范围是如何确定的？下面我们来回答这些问题（以 8 位数为例），原码不能直接运算，并且 0 的原码有＋0 和－0 两种形式，即 00000000 和 10000000，这样肯定是不行的，如果根据原码计算设计相应的门电路，由于要判断符号位，设计的复杂度会大大增加，不合算；为了解决原码不能直接运算的缺点，人们提出了反码的概念，但是 0 的反码还是有＋0 和－0 两种形式，即 00000000 和 11111111，这样是不行的，因为计算机在计算过程中，不能判断遇到 0 是＋0 还是－0；而补码解决了 0 表示的唯一性问题，即不会存在＋0 和－0，因为＋0 是 00000000，它的补码是 00000000，－0 是 10000000，它的反码是 11111111，再加 1 就得到其补码是 100000000，舍去溢出量就是 00000000。知道了计算机中数用补码表示和 0 的唯一性问题后，就可以确定数据类型表示的取值范围了，仍以字节类型为例，一个字节共 8 位，有 00000000～11111111 共 256 种结果，由于 1 位表示符号位，7 位表示数据位，正数的补码好说，其范围从 00000000～011111111，即 0～127；负数的补码为 10000000～11111111，其中，11111111 为－1 的补码，10000001 为－127 的补码，那么到底 10000000 表示什么最合适呢？8 位二进制数中，最小数的补码形式为 10000000；它的数值绝对值应该是各位取反再加 1，即为 01111111＋1＝10000000＝128，又因为是负数，所以是－128，即其取值范围是－128～127。

3. 可读性

图书的内容要深入浅出，使人爱看、易懂。一本书要做到可读性好，必须做到“善用比喻，实例为王”。什么是深入浅出？就是把复杂的事物简单地描述明白。把简单事情复杂化的是哲学家，而把复杂的问题简单化的是科学家。编写教材时要以科学家的眼光去编写，把难懂的定义，要通过图形或者举例进行解释，这样能达到事半功倍的效果。例如，在数据库中，第一范式、第二范式、第三范式、BC 范式的概念非常抽象，很难理解，但是，如果以一个教务系统中的学生表、课程表、教师表之间的关系为例进行讲解，从而引出范式的概念，学生会比较容易接受。再例如，在生物学中，如果纯粹地讲解各个器官的功能会比较乏味，但是如果提出一个问题，如人的体温为什么是 37℃？以此为引子引出各个器官的功能效果要好得多。再例如，在讲解数据结构课程时，由于定义多，表示抽象，这样达不到很好的教学效果，可以考虑在讲解数据结构及其操作时用程序给予实现，让学生看到直接的操作结果，如压栈和出栈操作，可以把 PUSH()和 POP()操作实现，这样效果会好



很多,并且会激发学生的学习兴趣。

4. 自学性

一本书如果适合自学学习,对其语言要求比较高。写作风格不能枯燥无味,让人看一眼就拒人千里之外,而应该是风趣、幽默,重要知识点多举实际应用的案例,说明它们在实际生活中的应用,应该有画龙点睛的说明和知识背景介绍,对其应用需要注意哪些问题等都要有提示等。

一书在手,从第一页开始的起点到最后一页的终点,如何使读者能快乐地阅读下去并获得知识?这是非常重要的问题。在数学上,两点之间的最短距离是直线。但在知识的传播中,使读者感到"阻力最小"的书才是好书。如同自然界中没有直流的河流一样,河水在重力的作用下一定沿着阻力最小的路径向前进。知识的传播与此相同,最有效的传播方式是传播起来损耗最小,阅读起来没有阻力。

欢迎老师投稿:bailj@tup.tsinghua.edu.cn。

2014 年 12 月 15 日

前　言

随着移动互联网时代的来临，智能手机、平板电脑、便携式导航等智能移动设备开始走入千家万户。越来越多的人开始把智能移动设备当作日常娱乐和办公的首选设备，随之而来的是移动平台下的应用软件开发需求日益旺盛，移动应用市场的前景不可估量。在众多智能移动设备操作系统中，Android 系统占据极其重要的地位，学习 Android 应用程序设计具有广阔的社会需求和实践意义。

作为一本面向初学者的教程，本书非常注重讲解的深入浅出和易学易懂，对于一些较难理解的理论，尽可能使用图示加以说明。对每个知识点都配有示例程序，并力求示例程序短小精悍，使其既能帮助读者理解知识，又具有启发性和实用性，非常适合教学讲授、自学或日后作为工具资料查询。每一章都配有难度适中的练习题，引导读者编写相关功能的实用程序，有助于提高学习兴趣。另外，为了帮助没有 Java 和 XML 基础的读者学习 Android 程序设计，本书特别设置了 Java 语言和 XML 的基础知识介绍，同时这部分内容还可以作为 Java 和 XML 语法简明手册使用，便于初学者在编程过程中查阅。

由于 Android 程序设计涉及编程语言、网络通信、硬件控制、多媒体等较多知识内容，所以学习时应该遵循循序渐进、由浅入深的原则，不可一蹴而就。学习的过程中既要注重理论的理解，更要强调动手实践，尤其对于初学者，多练习才能熟能生巧，才能掌握设计的方法和技巧。

本书共分 11 章。第 1 章介绍智能移动设备及其操作系统、Android 系统的体系结构，以及 Java、XML 等 Android 程序设计必要的预备知识。第 2 章介绍在 Windows 系统中搭建 Android 开发平台的主要步骤和集成开发环境的使用方法，并且通过学习创建第一个 Android 应用程序，了解典型 Android 应用程序的架构与组成。第 3 章介绍 Android 应用程序的一般开发流程和调试过程、调试工具和调试方法，以及应用程序的签名、打包和发布过程。第 4 章和第 5 章介绍用户界面的设计，主要包括 XML 布局文件的设计和使用方法、常见的界面布局方式、Android 中的事件处理机制，以及常用的用户界面控件。第 6 章介绍 Intent 的概念及其在组件通信中的应用，包括 Activity 之间跳转与通信、广播消息的发送和接收，以及 AppWidget 的相关概念和设计方法。第 7 章介绍 Android 系统的后台服务及其使用方法。第 8 章介绍 Android 常用的数据存储和访问方法，包括文件存储、SQLite 数据库存储、内容提供器(Content Provider)等。第 9 章介绍在 Android 系统中如何处理和使用图片、音视频等多媒体资源。第 10 章主要介绍访问 Internet 资源的方法，包括利用 HttpURLConnection、HttpClient 或 Socket 与远程服务器交互、使用 WebView 控件在 Activity 中包含一个基于 WebKit 的浏览器、通过使用

WebService 调用远程服务器上的方法等。第 11 章介绍几个综合应用的实例，通过学习这些实例，加深对基本知识的理解，提高对 Android 系统各个功能综合应用的能力。书中所有的示例程序均已在 Android 4.4.2（API Level 19）下调试通过，JDK 版本为 jdk-6u10-rc2-bin-b32-windows-i586，开发环境版本为 adt-bundle-windows-x86-20140321。

在本书的编写过程中，张雪梅负责编写第 1～6 章、第 10 章，高凯负责编写第 7～9 章、第 11 章，最后由高凯审阅全书。本书也得到河北省自然科学基金（No. F2013208105）、河北省高等学校科学技术研究重点项目（No. ZD2014029）的支持。读者可登录清华大学出版社网站（www.tup.com.cn）下载本书的全部源代码、电子课件和相关文件。

由于水平有限，书中难免会有不足之处，敬请读者批评指正！作者的联系方式是 zxm@hebust.edu.cn，欢迎来信交流，共同探讨 Android 程序设计方面的问题。

编　者

2015 年 1 月

目 录




XIII

第 1 章 Android程序设计起步

本章首先介绍智能移动设备及其操作系统以及 Android 系统的体系结构，然后介绍 Android 程序设计必要的预备知识，包括 Java 语言基础和 XML 的相关知识。

1.1 智能移动设备及其操作系统

智能移动设备一般像个人电脑一样具有独立操作系统和良好的用户界面，可由用户自行安装或删除应用程序。目前常见的用于智能移动设备的操作系统有 Android、Symbian、Windows Mobile、Windows Phone、BlackBerry 和 iOS 等。目前，这些操作系统之间的应用软件并不互相兼容。

Android 是一种以 Linux 为基础的开放源代码操作系统，最初主要支持手机，2005 年之后逐渐扩展到平板电脑及其他领域上。

Symbian 操作系统是一款面世较早的手机操作系统，曾广泛应用于诺基亚、摩托罗拉等主流机型，是手机领域中应用范围较广的操作系统之一。Symbian 拥有相当多针对不同用户的界面，它最大的特点就是采用了系统内核与人机界面分离技术，操作系统通常会因为手机的具体硬件而作改变，在不同的手机上它的界面和运行方式都有所不同。Symbian 对于硬件的要求比较低，支持多种语言环境，兼容性和扩展性非常出色。

Windows Mobile 是微软公司为智能移动终端设备开发的操作系统，将用户熟悉的桌面 Windows 体验扩展到了移动设备上。Windows Mobile 作为微软公司的掌上版本操作系统，在与桌面 PC 和 Office 办公软件的兼容性方面具有先天的优势，而且具有强大的多媒体性能，办公娱乐两不误。但其软件使用复杂、系统不太稳定、硬件要求较高。

Windows Phone 是微软公司发布的一款针对智能手机的操作系统。Windows Phone 具有桌面定制、图标拖拽、滑动控制等功能，其主屏幕通过提供类似仪表盘的体验来显示新的电子邮件、短信、未接来电、日历约会等，让人们对重要信息保持时刻更新。它还包括一个增强的触摸屏界面，以及一个 IE Mobile 浏览器。

BlackBerry 是 RIM 公司的产品。RIM 进入移动市场的时间比较早，并且开发出了适应美国市场的邮件系统，所以在美国市场的占有率很高。但是由于其定位于商务机，所以在多媒体播放方面的功能相对较弱。BlackBerry 在美国之外的影响非常小，市场占有率也较低。

iOS 操作系统的原名为 iPhone OS，是苹果公司的产品，主要用于 iPhone 和 iPod touch，其最大优势是操作过程具有出色的体验感。

随着智能手机应用的普及，各大手机平台也都推出了用于开发手机软件的 SDK (Software Development Kit)，如苹果公司推出了 iPhone 的 SDK，谷歌公司推出了 Android 的 SDK，Symbian 推出了 S60 SDK 等。SDK 大大降低了开发智能手机软件的门槛。但由于手机有着和普通 PC 不一样的特点，开发和运行过程中需要考虑屏幕大小、内存大小、背景色、省电模式的使用，以及实际的操作特点等因素，因此开发智能手机应用软件也有着和开发普通计算机应用程序不一样的特点。

在上述众多智能移动设备操作系统中，Android 系统占据极其重要的地位。与 Windows Mobile、iOS 这些专有操作系统不同，Android 通过提供一个以开源的 Linux 内核为基础而构建的、开放的开发环境，为移动应用程序的开发提供了新机遇。

目前使用 Android 系统的移动设备，尤其是 Android 手机在全球的市场具有越来越高的占有率，学习 Android 系统，开发基于 Android 的应用程序，逐渐成为当前应用开发的一个热点。本书重点介绍 Android 系统的特点和应用软件开发方法。

1.2 Android 系统的体系结构

1.2.1 Android 系统简介

Android 一词的本义指"机器人"，它是 Google 公司 2007 年 11 月推出的基于 Linux 平台的开源手机操作系统，其 Logo 如图 1-1 所示。Android 系统由底层 Linux 操作系统、中间件(负责硬件和应用程序之间的沟通)、核心应用程序组成，同时它也是一个免费、开放的智能移动设备开发平台。除了操作系统和用户界面，Google 公司还开发了手机地图、Gmail 等一些专用于 Android 手机的应用。目前 Android 系统已经逐渐发展成为最流行的手机和平板设备的操作系统和开发平台。



图 1-1 Android 系统的 Logo

Google 公司在 2007 年 11 月发布 Android 1.0 的同时，宣布成立了开放手机联盟。开放手机联盟由 Google 与三十多家移动技术和无线应用的领军企业组成，包括手机和终端制造商、芯片厂商、软件公司、移动运营商等。开放手机联盟旨在普及 Android 智能手机，负责推广和制造 Android 手机，支持更新和完善 Android 操作系统，使得 Android 能更好地发展。

2008 年 9 月 22 日，美国运营商 T-Mobile USA 在纽约正式发布第一款 Google 手机——T-Mobile G1。该款手机为宏达电子制造，是世界上第一部使用 Android 操作系统的手机，支持 WCDMA/HSPA 网络，理论下载速率为 7.2Mbps，并支持 Wi-Fi。

Android 是一个运行在 Linux 内核上的轻量级操作系统，功能全面，包括一系列 Google 公司在其内置的应用软件，如电话、短信等基本应用功能。Android 系统提供了开



放的 Android SDK 软件开发组件，它方便了开发人员开发 Android 应用程序。一般地，用户可以使用 Java 语言来开发 Android 平台上的应用程序，并通过 Android SDK 提供的一些工具将其打包为 Android 平台使用的 APK 文件，再使用模拟器或直接将其安装到 Android 移动设备上测试软件，检查软件实际运行情况和效果。

由于 Google 公司与开放手机联盟建立了战略合作关系，建立了标准化、开放式的通信软件平台，所以只要采用 Android 操作系统的平台，基本不受限于硬件设备，应用程序的可移植性好，能很好地解决当前由于众多手机操作系统的不同而造成的智能移动设备之间文件格式不兼容和信息无法互相流通的问题。

此外，Android 拥有完善的程序开发环境，如设备模拟器、调试工具、内存和性能分析工具等，应用程序框架可以方便地重用。Android 系统采用了处理速度更快的 Dalvik 虚拟机，集成了基于开源 WebKit 引擎的浏览器以及轻量级数据库管理系统 SQLite，拥有优化的图形系统和自定义的 2D/3D 图形库，支持常见的音频和视频以及各种图片格式。在相应硬件支持下，可集成 GSM、蓝牙、3G、Wi-Fi、摄像头、GPS、罗盘、加速度计等，这些硬件环境目前多数智能移动设备都能够提供。

图 1-2 展示了 Android 的设备模拟器，模拟设备是 3.2″HVGA Slider(ADPI)，从中可以初步了解 Android 的运行界面。



图 1-2 Android 的设备模拟器

1.2.2 Android 系统的总体架构

Android 系统由底层 Linux 操作系统、中间层的中间件、上层 Java 应用程序组成，其总体架构如图 1-3 所示。Android 系统的体系结构分为 4 层，从下到上依次为 Linux 内核、核心类库和 Android 运行时、应用程序框架、应用程序。

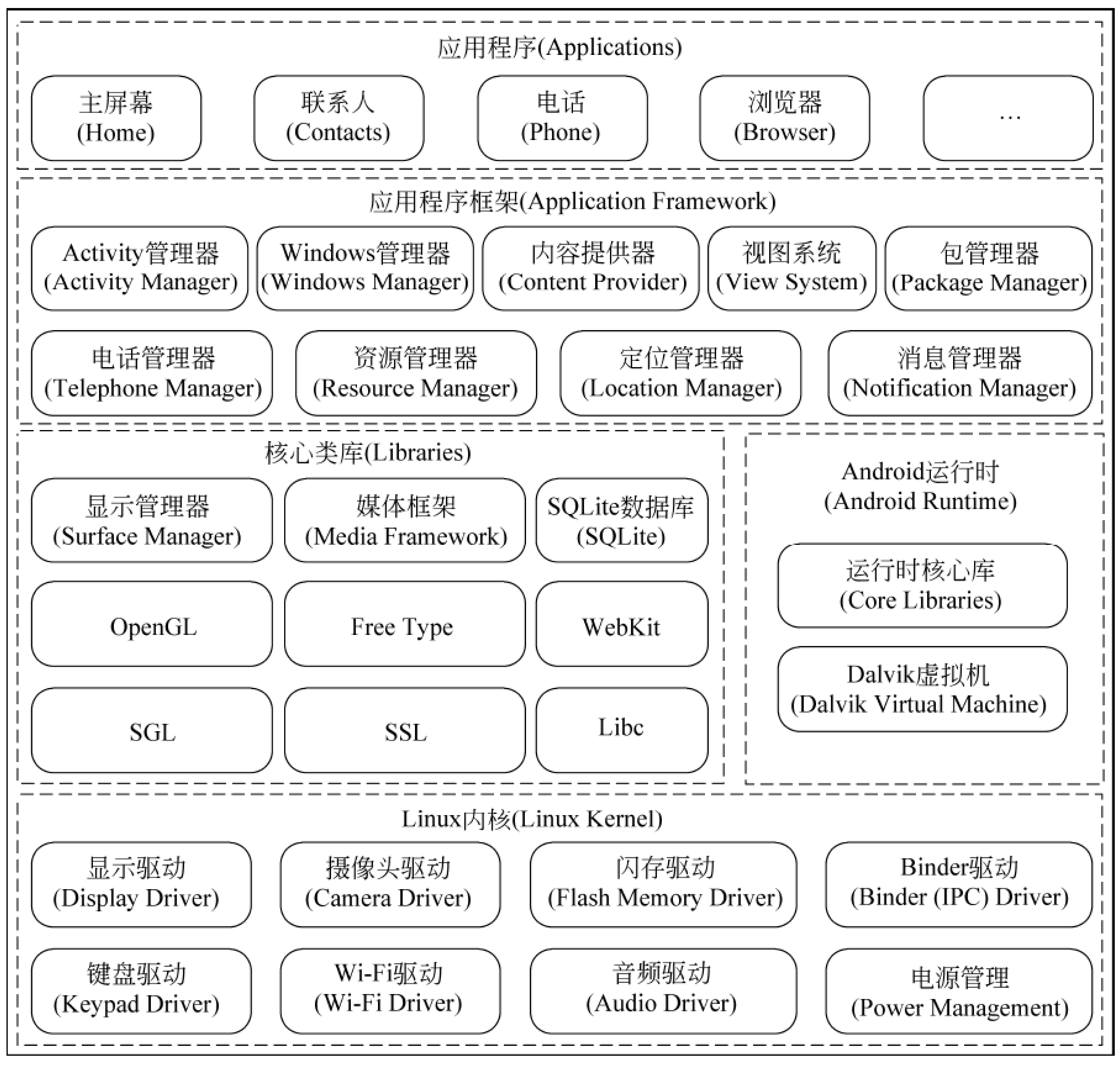


图 1-3 Android 系统的总体架构

1. Linux 内核

Android 系统的最底层是基于 Linux 内核(Linux Kernel)实现的,它负责硬件驱动、网络管理、电源管理、系统安全、内存管理等。例如,它可以负责显示驱动、基于 Linux 的帧缓冲驱动、键盘驱动、Flash 驱动、摄像头驱动、音频视频驱动和 Wi-Fi 驱动等。

2. Android 核心类库

Android 系统的第二层由核心类库(Libraries)和 Android 运行时(Android Runtime)组成。核心类库包括开源的函数库,如标准的 C 函数库 Libc、OpenSSL、SQLite 等。其中 WebKit 是负责网页浏览器运行的类库;SGL/OpenGL 是 2D 和 3D 图形与多媒体函数库,分别支持各种影音与图形文件的播放与显示;SQLite 提供了轻量型的数据库管理系统。

3. Android 运行时

Android 运行时(Android Runtime)环境也位于框架第二层,提供了 Android 特有的 Java 内核函数库。另外,Android 为每个应用程序分配了专有的 Dalvik 虚拟机,可以通过 Java 语言编写应用程序并在 Android 平台上同时运行多个 Java 应用程序。Dalvik 虚拟机对有限内存、电池和 CPU 进行了优化,处理速度更快,同时拥有可在一个设备上运行多个虚拟机的特性。Dalvik 虚拟机运行的.dex 格式文件经过了优化,占用的内存非常小,执行效率非常高。

4. 应用程序框架

Android 系统的第三层是应用程序框架(Application Framework),它为应用程序层的开发者提供用于软件开发的 API。由于最上层的应用程序是以 Java 构建的,因此该层提供的组件包含了用户界面 UI 中所需要的各种控件。相应功能有显示(如文字、条列消息、按钮、内嵌式浏览器等)、消息提供(如访问信息、分享信息)、资源管理(如图形、布局文件等)、提示消息(如显示警告信息)等。例如,框架中的 Activity Manager 负责在设备上生成窗口事件,而 View System 则在窗口显示设定的内容。

5. 应用程序

Android 系统的最上层是应用程序(Applications)。Android 系统本身已经提供了一些核心的应用,如主屏幕、联系人、电话、浏览器、游戏,以及 Google Maps、E-mail、即时通信工具、MP3 播放器、电话、照相程序、文件管理等。同时,开发者还可以使用 SDK 提供的 API 开发自己的应用程序。本书的重点就是介绍如何使用 SDK 提供的 API 开发自己的应用程序。

Android 应用程序一般使用 Java 作为开发语言,但不是由传统的 Java 虚拟机运行,而是转换为.dex 文件格式后,由 Dalvik 虚拟机运行。Dalvik 虚拟机和一般 Java 虚拟机有所不同,它执行的不是 Java 标准的字节码,而是.dex 格式的可执行文件。与普通的 Java 虚拟机基于栈不同,Dalvik 虚拟机是基于寄存器的,其好处在于可以实现更多的优化,这更适合移动设备的特点。

在开发 Android 应用程序的过程中通常采用 Eclipse 作为编程 IDE,本书即以此作为开发环境介绍。

总之,Android 采用了开源的 Linux 操作系统,底层使用了硬件访问速度最快的 C 语言,应用层采用了简单又强大的 Java 语言,博采众长,使其具有无限的魅力和生命力,受到业界的极大欢迎。

1.2.3 Android SDK 简介

Android SDK 提供了在 Windows/Linux/Mac 平台上开发 Android 应用程序的开发组件,它含有在 Android 平台上开发应用程序的工具集。Android SDK 包含了大量的类库和开发工具,程序开发者可以直接调用这些 API 函数。

Android SDK 提供的开发工具包括用于在 Eclipse 中的开发工具插件 ADT、调试工具、内存和性能分析工具、打包成 APK 文件的工具、用于模拟和测试软件的虚拟设备 AVD、Dalvik 虚拟机、基于开源 WebKit 引擎的浏览器、2D/3D 图形界面、轻量级数据库管理系统 SQLite，以及对摄像头、GPS、Wi-Fi 等硬件的支持。Android SDK 支持常见的音视频和各种图片格式。

与普通 Java 程序运行时需要的 JRE 运行环境不同，Android 通过 Dalvik 而非直接采用 Java 的虚拟机来运行 Android 程序。Dalvik 虚拟机针对移动设备的实际情况进行了功能优化，如支持多进程与内存管理、低功耗支持等。和普通 Java 虚拟机不同的是，Dalvik 支持运行的文件格式是特殊的，因此它需要将普通 Java 的 class 文件用 Android SDK 中的 dx 工具转换为.dex 格式的文件，这些转换对使用 Eclipse ADT 的程序开发者而言是透明的，无须编程人员处理。

Android SDK 中的各种相关包被组织成 android.* 的方式。例如 android.app 包提供程序模型、基本的运行环境，如 Activity、ListActivity 等；android.widget 包提供各种 UI 元素，如 TextView、Button、ListView 等；android.content 包提供对数据进行访问和发布的类，如 ContentProvider、Intent 等；android.graphics 包提供底层的图形、服务，如 Canvas、Cursor 等。要在自己的程序中使用这些包中的类，必须先用 import 语句引入相关包文件，如在编程时如果需要使用颜色相关类，则引入 android.graphics.Color 包，使用不同的字体，则引入 android.graphics.Typeface 包。

1.3 Java 语言与面向对象编程基础

Android 应用程序一般使用 Java 作为开发语言，Android 应用开发水平的高低很大程度上取决于 Java 语言能力，所以在学习 Android 应用设计之前要了解 Java 语言与面向对象编程方法。

Java 是一种可以编写跨平台应用软件的面向对象程序设计语言，是由 Sun Microsystems 公司于 1995 年 5 月推出的。Java 具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于 PC、数据处理、游戏控制、科学计算、移动电话和互联网等领域。Java 的语言风格十分接近 C 和 C++。它继承了 C++ 语言面向对象技术的核心，提供类、接口和继承等原语，但舍弃了 C++ 语言中容易引起错误的指针、运算符重载和多重继承等特性。

Java 语言是一个纯粹的面向对象的程序设计语言，其全部设计工作都集中于对象及其接口。对象中封装了它的状态变量以及相应的方法，实现了模块化和信息隐藏。而类则提供了一类对象的原型，并且通过继承机制，子类可以使用父类所提供的方法，实现了代码的复用。

Java 提供了大量的类以满足网络化、多线程、面向对象系统的需要。这些类被分别放在不同的包中，供应用程序使用。例如，语言包提供字符串处理、多线程处理、异常处理、数学函数处理等类，实用程序包提供的支持包括哈希表、堆栈、可变数组、时间和日期等，抽象图形用户接口包实现了不同平台的计算机的图形用户接口部件，包括窗口、菜单、



滚动条、对话框等。

1.3.1　配置 Java 开发环境

在开始 Java 编程之前,需要安装 JDK(Java Development Kit),配置 Java 开发环境。JDK 是提供 Java 服务的系统包,用于开发和测试 Java 程序。安装和配置 JDK 环境变量的步骤如下。

步骤 1:下载 Java 开发环境工具包。

进入网址 http://www.oracle.com/technetwork/java/javase/downloads/index.html 对应的网页。单击“下载 JDK”链接,下载工具包软件。网页会显示一个许可协议,单击“接受许可协议”,就会看到一系列安装文件的下载链接,如图 1-4 所示。选择自己的操作系统,然后将文件保存到本地目录中。

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u5-linux-i586.rpm
Linux x86	152.5 MB	jdk-8u5-linux-i586.tar.gz
Linux x64	133.87 MB	jdk-8u5-linux-x64.rpm
Linux x64	151.64 MB	jdk-8u5-linux-x64.tar.gz
Mac OS X x64	207.79 MB	jdk-8u5-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.68 MB	jdk-8u5-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	95.54 MB	jdk-8u5-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.9 MB	jdk-8u5-solaris-x64.tar.Z
Solaris x64	93.19 MB	jdk-8u5-solaris-x64.tar.gz
Windows x86	151.71 MB	jdk-8u5-windows-i586.exe
Windows x64	155.18 MB	jdk-8u5-windows-x64.exe

图 1-4　下载 JDK 安装文件

步骤 2:安装开发工具包。

运行步骤 1 下载的.exe 文件,文件将自动解压并安装开发工具包。

安装完 JDK 后,在安装目录下会增加很多目录和文件。其中 bin 文件夹中是 JDK 的基本程序和工具;jre 文件夹中是 Java 运行时的环境;lib 文件夹中是 Java 类库;Demo 文件夹中存放 Java 自带的一些示例程序。

JDK 的帮助文件有在线版本和离线版本两种,可以从 Java 的官方网站上下载。帮助文件分为两种格式,有 HTML 格式和 CHM 格式。只需要打开目录下的 index.html 即可使用 JDK 的帮助文件,根据包的路径可以查找到所有的类、属性和方法。

步骤 3:配置环境变量。

环境变量是指供系统内部使用的变量,是包含系统的当前用户的环境信息的字符串和软件的存放路径,安装完 JDK 后必须配置环境变量。

配置环境变量的方法是:右击 Windows 桌面的“我的电脑”图标,在弹出的快捷菜单中选择“属性”,然后在弹出的对话框中选择“高级”选项卡,单击其中的“环境变量”按钮,弹出“环境变量”对话框,如图 1-5 所示。

在“系统变量”栏中设置 3 项属性:JAVA_HOME、PATH、CLASSPATH。若这些变量已存在,则单击“编辑”按钮,在原值基础上添加新变量的值,原值和新值之间用分号间隔;否则单击“新建”按钮,添加变量名和变量值。变量名称和值不区分大小写。

图 1-5 “环境变量”对话框

JAVA_HOME 变量值用于指明 JDK 的安装路径，就是前述安装 JDK 时所选择的路径，如 D:\Java\jdk1.6.0_10，此路径下包括 lib、bin、jre 等文件夹。运行 Tomcat、Eclipse 等都需要使用此变量。

PATH 变量值使得系统可以在任何路径下识别 Java 命令，其值设为“%JAVA_HOME%\bin; %JAVA_HOME%\jre\bin”。

CLASSPATH 变量值是 Java 加载类(class 或 lib)的路径，只有类在 CLASSPATH 中，Java 命令才能识别，其值设为“.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar”。

设置完成后，执行 Windows 的“开始”→“运行”命令，在“运行”对话框中输入 cmd 命令，按 Enter 键后则进入命令提示符窗口。在窗口中输入 java-version、java、javac 等 JDK 命令，如果能正常运行，如图 1-6 所示，说明环境变量配置正确，这时可以编写并执行 Java 程序。

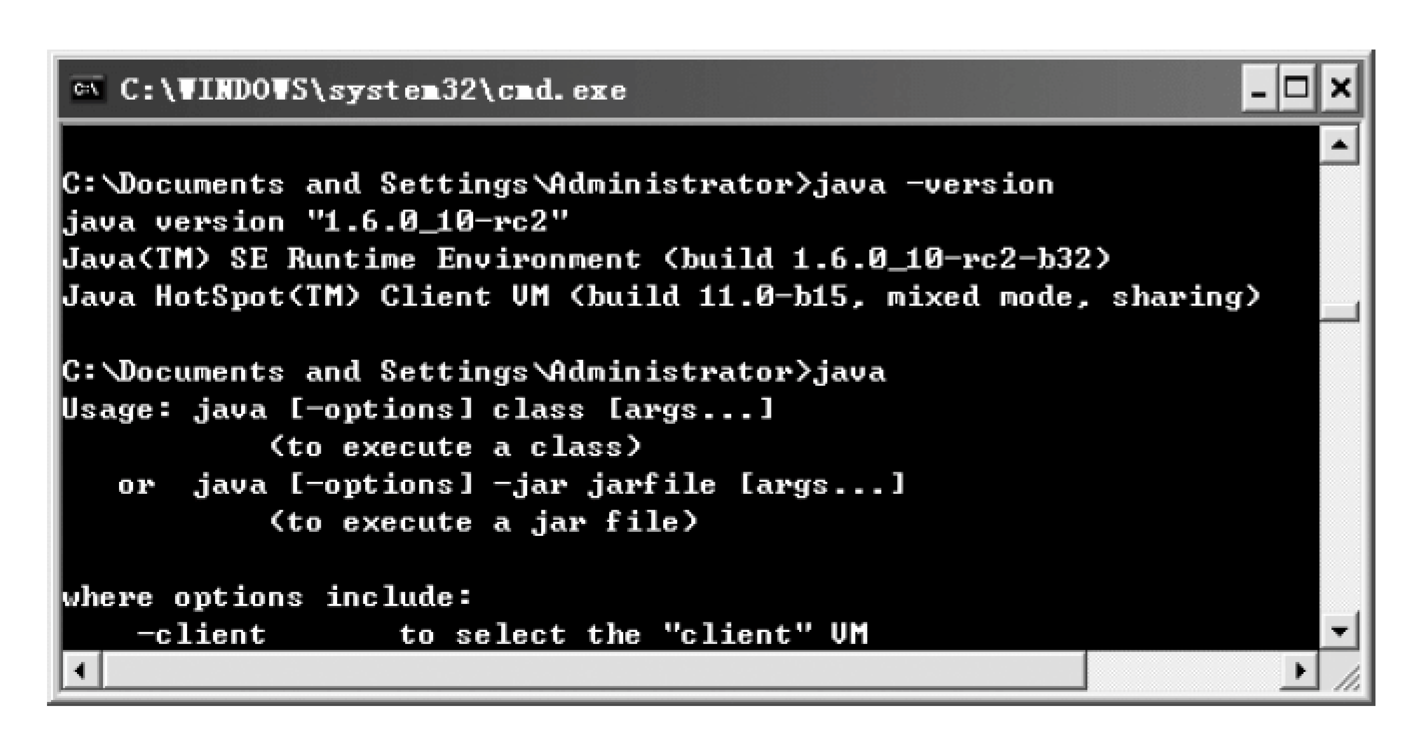


图 1-6 运行 JDK 命令

1.3.2　Java 程序的开发过程

Java 不同于一般的编译语言或解释语言。它首先将源代码编译成二进制字节码(bytecode)，然后依赖各种不同平台上的虚拟机来解释执行字节码。从而实现了“一次编译、到处执行”的跨平台特性。

编辑 Java 源代码可以使用任何无格式的纯文本编辑器，如 Windows 操作系统上的记事本，也可使用更高级的编程工具，如 Eclipse、JBulder、NetBeans 等，这些工具具有更加强大的辅助功能。

Eclipse 是目前最流行的 Java 编程工具之一，在 Eclipse 中集成了许多工具和插件，从而使 Java 程序的开发更容易。这是一款可以免费使用的软件，可以从 Eclipse 的官方网站(http://www.eclipse.org/)下载，解压后无须安装，运行其中的 eclipse.exe 文件就可以使用。

安装好 JDK 及配置好环境变量以后，就可以进行 Java 程序的开发。开发过程要经过以下 3 个步骤。

步骤 1：创建一个源文件。Java 源文件就是 Java 代码文件，以 Java 语言编写。Java 源文件是纯文本文件，扩展名为 java。

如果使用 Eclipse 作为 Java 编程环境，通常先创建一个 Java 工程项目，然后在项目中创建 Java 源文件。

步骤 2：将源文件编译为一个.class 文件。使用 JDK 所带的编译器工具 javac.exe，它会读取源文件并将其文本编译为 Java 虚拟机能理解的指令，保存在以后缀.class 为结尾的文件中。包含在 class 文件中的指令就是字节码(bytecodes)，它是与平台无关的二进制文件，执行时由解释器 java.exe 解释成本地机器码，边解释边执行。

步骤 3：运行程序。使用 Java 解释器(java.exe)来解释执行 Java 应用程序的字节码文件(.class 文件)，通过使用 Java 虚拟机来运行 Java 应用程序。

如果使用 Eclipse 作为 Java 编程环境，前述步骤 2 和步骤 3 可以由 Eclipse 自动完成。

1.3.3　Java 程序的结构

Java 应用程序分为 Application 与 Applet 两种，它们有不同的程序结构和运行方式。

下面的示例分别在 Eclipse 环境中编写了一个 Application 程序和一个 Applet 程序，并编译和运行了程序。

【例 1-1】 工程项目 01_HelloWorld 演示了一个 Application 程序，其功能是在控制台输出字符串“HelloWorld!”。

在 Eclipse 中新建一个 Java 工程项目，项目名称为 01_HelloWorld。创建完成后在 Eclipse 左侧的 Package Explorer 面板中会看到工程项目的树型结构，如图 1-7 所示。其中 src 文件夹用于存放 Java 源代码文件。



图 1-7　Package Explorer 面板

新建类 HelloWorldApp,内容如下。

```
public class HelloWorldApp {
    public static void main(String args[]) {
        System.out.println("HelloWorld!");
    }
}
```

该程序的运行结果是在控制台输出下面一行信息:

```
HelloWorld!
```

该程序中,首先用保留字 class 来声明一个新的类,其类名为 HelloWorldApp,它是一个公共类(public)。整个类定义由大括号{}括起来。在该类中定义了一个 main()方法,其中 public 表示访问权限,指明所有的类都可以使用这一方法;static 指明该方法是一个静态方法,它可以通过类名直接调用;void 则指明 main()方法不返回任何值。

对于一个 Application 程序来说,main()方法是必需的,而且必须按照如上的格式来定义。Java 解释器在没有生成任何实例的情况下,以 main()作为入口来执行程序。一个 Java 程序中可以定义多个类,每个类中可以定义多个方法,但是最多只能有一个公共类,main()方法也只能有一个,作为程序的入口。

在 main()方法定义中,括号中的 String args[]是传递给 main()方法的参数。参数名为 args,它是类 String 的一个实例,参数可以为 0 个或多个,多个参数间用逗号分隔。

在本例中,main()方法的实现只有一条语句,它用来实现将字符串输出到控制台。

运行该程序时,首先把它保存成一个名为 HelloWorldApp.java 的文件,文件名必须和类名相同。编译的结果是生成字节码文件 HelloWorldApp.class。

【例 1-2】 工程项目 01_AppletExample 演示了一个 Applet 程序,其功能是输出字符串“HelloWorld!”。

在 Eclipse 中新建一个 Java 工程项目,项目名称为 01_AppletExample。新建类 HelloWorldApplet,内容如下。

```
import java.awt.* ;
import java.applet.* ;
public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("HelloWorld!",20,20);
    }
}
```

这是一个简单的 Applet 小程序。程序中,首先用 import 语句引入 java.awt 和 java.applet 下所有的包,使得该程序能够使用这些包中所定义的类。然后声明一个公共类 HelloWorldApplet,用 extends 指明它是 Applet 的子类。在类中,重写父类 Applet 的 paint()方法,其中参数 g 为 Graphics 类,它表明当前绘制的上下文。在 paint()方法中,调用 g 的 drawString()方法,在坐标(20,20)处输出字符串“HelloWorld!”。绘制时,坐标



原点位于显示区域的左上角，正方向分别是向右和向下，坐标值是用像素点来表示的。

本例的运行结果是在屏幕上弹出一个 Applet Viewer 窗口，在其中的指定坐标处显示字符串“HelloWorld!”，如图 1-8 所示。



图 1-8　Applet Viewer 中的运行结果

这个程序中没有定义 main()方法，这是 Applet 与 Application 的区别之一。为了运行该程序，首先也要把它存储成文件 HelloWorldApplet.java，然后对它进行编译，得到字节码文件 HelloWorldApplet.class。由于 Applet 中没有 main()方法作为 Java 解释器的入口，必须编写 HTML 文件，把该 Applet 嵌入其中，在支持 Applet 的浏览器中运行，或用 Applet Viewer 来运行。

嵌入 Applet 的 HTML 文件如下。

```
<HTML>
  <HEAD>
    <TITLE> An Applet</TITLE>
  </HEAD>
  <BODY>
    <applet code="HelloWorldApplet.class" width=200 height=40>
    </applet>
  </BODY>
</HTML>
```

从上述例子中可以看出，Java 程序是由类构成的，对于一个应用程序来说，必须有一个类中定义 main()方法，而对 Applet 小程序来说，它必须作为 Applet 的一个子类。在类的定义中，应包含类变量的声明和类中方法的实现。

1.3.4　Java 的数据类型和运算符

Java 是一个强类型的语言，要求在使用变量前必须显式定义变量并声明变量值的类型。数据类型指明了变量或表达式的状态和行为。

1. 数据类型

Java 不支持 C、C++ 中的指针类型、结构体类型和共用体类型。

Java 中的数据类型分为基本数据类型和引用数据类型两大类。Java 支持的简单类型有整型、实型、布尔型和字符型，其基本数据类型一共有 8 种，byte、char(字符型，表示一个字符，常量用单引号来表示)、int(整型)、short、long、float(单精度浮点类型)、double(双精度浮点类型)、boolean(布尔型)。引用数据类型包括数组、类(包括对象)和接口。

在 Java 中，有一些数据类型之间是能够进行数据类型转换的。转换方式有自动转换和强制转换两种。自动转换就是不需要明确指出所要转换的类型是什么，是由 Java 虚拟机自动来转换的。转换的规则一般是小数据类型转换为大数据类型，但大的数据类型的

数据精度有的时候要被破坏。对于引用类型，子类类型可自动隐式转换为父类类型。

把一个能表示更大范围或者更高精度的类型，转换为一个范围更小或者精度更低的类型时，就需要使用强制类型转换。所谓强制转换，是指在程序中显式控制的一种强制性类型转换，例如：

```
int a=25;                    //定义数据类型,a为 int类型变量
long b=133;                  //定义数据类型,b为 long类型变量
char c=(char)a;              //强制转换数据类型,将 a强制转换成 char类型
int n=(int)b;                //强制转换数据类型,将 b强制转换成 int类型
```

但要注意，当大数据类型转换成小数据类型时，强制类型转换有可能会造成溢出或丢失精度，使数值发生变化。如上例中的(int)b，b 原来是 long 型，要将它强制转换成 int 整型，转化后的数值就有可能发生变化。

2. 标识符

在 Java 里，方法名、类名、变量名都是标识符。标识符必须以英文字母开头，是由英文字母或数字组成的，其他的符号不能出现在标识符里。其中英文字母包括大写的 A～Z，小写的 a～z，以及“_”和 $；数字包括 0～9。标识符不能使用 Java 保留的关键字。特别要注意的是，在 Java 里标识符是大小写敏感的。

符合标识符的命名规则并不一定是一种最好的命名方法。给一个标识符命名首先要符合命名规范，最好见名知意。

3. 变量

变量是程序中的基本存储单元，它的定义包括变量名、变量类型和作用域几部分。Java 变量名必须是一个合法的标识符，不能以数字开头。声明一个变量的同时也指明了变量的作用域。例如，类中声明变量，而不是在类的某个方法中声明，则它的作用域是整个类；方法定义中的形式参数用于给方法内部传递数据，则它的作用域就是这个方法。

只有局部变量和类变量是可以赋初值的，而方法参数和例外处理参数的变量值是由调用者给出的。

变量的声明格式如下：

```
type identifier[=value][,identifier[=value]…];
```

例如：

```
int a,b,c;
double d1,d2=0.0;
```

4. 常量

Java 中的常量值是用字符串表示的，它区分为不同的类型。如整型常量 123、实型常量 1.23、字符型常量'a'、布尔型常量 true、false 以及字符串常量"helloWorld."。Java 中用



关键字 final 来实现把一个标识符定义为常量，例如：

```
final double PI=3.1415926;
```

在 Java 中，对于用 final 限定的常量，在程序中不能改变它的值。通常常量名全部使用大写字母。

5. 运算符

运算符指明对操作数所进行的运算。Java 支持的运算符包括算术运算符、关系运算符、布尔逻辑运算符、位运算符、赋值运算符和条件运算符等，具体如表 1-1 所示。运算符的运算优先级是有一定的顺序的，括号拥有最高的优先级，接下来依次是一元运算符和二元运算符。

表 1-1　Java 支持的运算符

分类	符号	说　明	分类	符号	说　明
算术运算符	+	加	关系运算符	>>	带符号右移
	－	减		<<	带符号左移
	*	乘		>>>	无符号右移
	/	除(整数除时商只取整数部分)		&	按位与
	%	求余		|	按位或
	++	自增，例如：i++相当于 i=i+1		^	按位异或
	－－	自减，例如：i－－相当于 i=i－1		～	按位取反
位运算符	>	大于	逻辑运算符	!	非
	<	小于		&&	与
	>=	大于等于		||	或
	<=	小于等于	条件运算符	?:	表达式 1? 表达式 2:表达式 3
	==	等于			
	!=	不等于			
赋值运算符	=	赋值	其他	( )	方法调用运算符
	+= －= *= 等	扩展赋值运算符，先运算再赋值		[ ]	下标运算符
				new	内存分配运算符
				(类型)	强制类型转换运算符
				.	点运算符

1.3.5 Java 的流程控制语句

Java 中的流程控制语句包括分支语句和循环语句。

1. 分支语句

分支语句有 if-else 语句、else if 语句和 switch 语句 3 种。

if-else 语句根据判定条件的真假来执行两种操作中的一种，语法格式如下：

```
if(boolean-expression) {
    statement1;
  }
[else {
    statement2;
  }]
```

else if 语句是 if-else 语句的一种特殊形式，语法格式如下：

```
if(expression1) {
    statement1
  }else if(expression2) {
    statement2
  }
  ⋮
  else if(expressionM) {
    statementM
  }else{
    statementN
}
```

switch 语句根据表达式的值来执行多个操作中的一个，语法格式如下：

```
switch(expression) {
    case value1: statement1;
                 break;
    case value2: statement2;
                 break;
    ⋮
    case valueN: statementN;
                 break;
    [default:defaultStatement;]
}
```

表达式 expression 可以返回任一简单类型的值(如整型、实型、字符型)，多分支语句把表达式返回的值与每个 case 子句中的值相比。如果匹配成功，则执行该 case 子句后的语句序列。

case 子句中的 value 值必须是常量，而且所有 case 子句中的值必须是不同的。

default 子句是任选的。当表达式的值与任一 case 子句中的值都不匹配时，程序执行 default 后面的语句。如果表达式的值与任一 case 子句中的值都不匹配且没有 default 子



句，则程序不执行任何操作，直接跳出 switch 语句。

break 语句用来在执行完一个 case 分支后，使程序跳出 switch 语句，即终止 switch 语句的执行。因为 case 子句只是起到一个标号的作用，用来查找匹配的入口，从此处开始执行，对后面的 case 子句不再进行匹配，而是直接执行其后的语句序列，因此在每个 case 分支后，要用 break 语句来终止后面的 case 分支语句的执行。在一些特殊情况下，多个不同的 case 值需要执行一组相同的操作，这时可以不用 break 语句。

switch 语句的功能可以用 else if 语句来实现，但在某些情况下，使用 switch 语句更简练，可读性强，而且程序的执行效率更高。

2. 循环语句

Java 的循环语句有 while 语句、do-while 语句和 for 语句 3 种。

while 语句实现"当型"循环，它的一般格式如下：

```
while(termination) {
    bodystatements;
}
```

当布尔表达式 termination 的值为 true 时，循环执行大括号中的语句。while 语句首先计算终止条件，当条件满足时，才去执行循环中的语句，这是"当型"循环的特点。

do-while 语句实现"直到型"循环，它的一般格式如下：

```
do{
    bodystatements;
}while(termination);
```

do-while 语句首先执行循环体，然后计算终止条件 termination，若结果为 true，则循环执行大括号中的语句，直到布尔表达式 termination 的结果为 false。与 while 语句不同的是，do-while 语句的循环体至少执行一次，这是"直到型"循环的特点。

for 语句也用来实现"当型"循环，它的一般格式如下：

```
for(initialization;termination;iteration) {
    bodystatements;
}
```

for 语句执行时，首先执行初始化操作 initialization，然后判断终止条件 termination 是否满足，如果满足，则执行循环体中的语句，最后执行迭代部分 iteration。完成一次循环后，重新判断终止条件 termination。

for 语句通常用来执行循环次数确定的情况，如对数组元素的操作，当然也可以根据循环结束条件执行循环次数不确定的情况。

1.3.6　数组

数组是一种存放多个相同类型数据的数据结构。

1. 一维数组的定义

一维数组的定义格式如下：

```
type arrayName[]=new type[arraySize];
```

其中 type(类型)可以为 Java 中任意的数据类型，数组名 arrayName 必须是一个合法的 Java 标识符，[]指明该变量是一个数组类型变量，arraySize 指明数组的长度，即数组元素的个数。例如：

```
int myArray[]=new int[3];
```

该语句声明了一个名为 myArray 的整型数组，数组中的每个元素为整型数据。用运算符 new 为它分配内存空间，本例分配了 3 个 int 型整数所需要的内存空间。

定义了一个数组，并用运算符 new 为它分配了内存空间后，就可以引用数组中的每一个元素了。数组元素的引用方式如下：

```
arrayName[index]
```

其中 index 为数组下标，它可以是整型常数或表达式。下标从 0 开始，一直到数组的长度减 1。对于上面例子中的 myArray 数组来说，它有 3 个元素，分别为 myArray[0]、myArray[1]和 myArray[2]。

可以单独对每个数组元素进行赋值，赋值方法与变量相同。也可以在定义数组的同时进行初始化，例如：

```
int myArray[]={1,2,3,4,5};
```

用逗号分隔数组的各个元素，系统自动为数组分配一定的空间。

2. 多维数组的定义

与 C、C++ 一样，Java 把多维数组看作数组的数组。例如二维数组为一个特殊的一维数组，其每个元素又是一个一维数组。

二维数组的定义方式如下：

```
type arrayName[][]=new type[arraySize1][arraySize2];
```

例如，下面的语句定义了一个 2×3 的整型数组。

```
int myArray[][]=new int[2][3];
```

对于二维数组中的每个元素，引用方式为 arrayName[index1][index2]，其中 index1、index2 为下标，可为整型常数或表达式，如 a[2][3]。与一维数组类似，每一维的下标都从 0 开始。

二维数组也可以在定义数组的同时进行初始化。例如，以下语句定义了一个 3×2 的数组，并对每个元素赋值：

```
int myArray[][]={{2,3},{1,5},{3,4}};
```



3. 动态数组列表 ArrayList

ArrayList 是一个类，定义在 java. util 包中。利用 ArrayList 可以定义一个可自动调节大小的数组。它最大的优点是可以自动改变数组的大小，灵活地插入元素和删除元素，但与普通数组相比，其执行速度要差一些。

使用 ArrayList 要首先创建一个 ArrayList 对象，例如，下面的语句创建了对象 myList：

```
ArrayList myList=new ArrayList();
```

之后就可以调用 add()方法为 ArrayList 对象数组增加元素，例如，下面的语句为 List 对象增加了一个 int 型元素，元素值为 12：

```
myList.add(12);
```

调用 RemoveAt()方法移除 ArrayList 对象的元素，例如：

```
myList.RemoveAt(5);                    //将第 6个元素移除
```

另外，调用 addAll()方法可以添加一批元素到当前列表的末尾，remove()方法可以删除一个元素，removeAll()方法可以删除所有元素，clear()方法可以清除现有所有的元素，toArray()方法可以把 ArrayList 的元素复制到一个数组中。

1.3.7　面向对象的编程方法

面向对象程序设计(Object Oriented Programming，OOP)是当前主流的程序设计方法。面向对象的程序设计方法按照现实世界的特点来管理复杂的事物，把它们抽象为对象。对象是由数据和对于这些数据的操作组成的封装体，与客观实体有直接对应关系。对象具有自己的状态和行为，通过对消息的响应来完成一定的任务。一个类定义了具有相似性质的一组对象。类具有继承性，这是对具有层次关系的类的属性和操作进行共享的一种方式。

面向对象编程过程简要来说分为以下几个步骤：首先分析要解决的问题，根据需求确定类及其属性；接着确定每个类的操作，这些操作都封装在类的方法中；然后使用继承机制来处理类之间的共同点；最后将这些类实例化成对象，实现程序的功能。

1. 基本概念

面向对象程序设计中涉及一些重要的概念，通过这些概念面向对象的思想得到了具体的体现。理解这些概念有助于人们运用面向对象的编程方法。

1) 对象

对象(Object)是要研究的任何事物。它不仅能表示有形的实体，也能表示抽象的规则、计划或事件。对象由数据(描述对象的属性)和作用于数据的操作(体现对象的行为)构成一个独立整体。从程序设计者的角度来看，对象是一个程序模块；从用户的角度来看，对象为他们提供所希望的行为。一个对象有状态、行为和标识 3 种属性。

在 Java 中，对象的属性称为“成员变量”，对象的行为称为“成员方法”或“成员函数”，一个对象就是变量和相关的方法的集合，其中变量表明对象的状态，方法表明对象所具有的行为。面向对象的程序设计实现了对象的封装，使人们不必关心对象的行为是如何实现的这样一些细节。通过对对象的封装，实现了模块化和信息隐藏，有利于程序的可移植性和安全性，同时也利于对复杂对象的处理。

2）消息

对象之间必须要进行交互来实现复杂的行为，交互是通过消息（Message）机制实现的。一个消息包含 3 个方面的内容：消息的接收者、接收对象应调用的方法、方法所需要的参数。同时，接收消息的对象在执行相应的方法后，可能会给发送消息的对象返回一些信息。

3）类

一个共享相同结构和行为的对象的集合称为“类”（Class）。通常来说，类定义了一类事物的共同属性和它们的行为。类是对象的模板，即类是对一组有相同数据和相同操作的对象的定义，一个类所包含的数据和方法描述了一组对象的共同属性和行为。类是在对象之上的抽象，对象则是类的具体化，是类的实例。

类可有其子类，形成类层次结构。它们之间具有继承性的关系，继承性是子类自动共享父类之数据和方法的机制。在这种关系中，一个类共享了一个或多个其他类定义的结构和行为。子类可以共享父类的成员变量和方法，同时可以对其进行扩展、覆盖和重定义，这样可以使子类有比父类更加强大的功能。

继承不仅支持系统的可重用性，而且还促进系统的可扩充性。在 Java 中通过接口可以实现多重继承。接口的概念简单，使用更方便，而且不仅仅限于继承，它使多个不相关的类可以具有相同的方法。

4）方法

方法（Method）也称为成员函数，是指对象上的操作，作为类声明的一部分来定义。方法定义了一个对象可以执行哪些操作。

在面向对象方法中，对象和传递消息分别表现事物及事物间相互联系的概念。这种基于对象、类、消息和方法的程序设计方法的基本点在于对象的封装性和类的继承性。通过封装能将对象的定义和对象的实现分开，通过继承能体现类与类之间的关系，以及由此带来的动态联编和实体的多态性，从而构成了面向对象的基本特征。

2. Java 中的编程方法

1）涉及的概念

抽象（abstract）类：包含一个或多个抽象方法的类。抽象类只是用来派生子类，而不能用它来创建对象。抽象是指在定义类的时候，确定了该类的一些行为和动作。例如，自行车可以移动，但怎么移动不进行说明，这种提前定义一些动作和行为的类称为抽象类。

final 类：它只能用来创建对象，而不能被继承，与抽象类刚好相反。abstract 与 final 不能同时修饰同一个类。

包：Java 中的包是相关类和接口的集合，创建包须使用关键字 package。

接口：Java 中的接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方



法的特征没有方法的实现，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为或功能。

重载：当多个方法具有相同的名字而含有不同的参数时，便发生了重载。编译器必须挑选出调用哪个方法进行编译。

重写：也可称为方法的“覆盖”。在 Java 中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想做一定的修改，这就需要采用方法的重写。值得注意的是，子类在重新定义父类已有的方法时，应保持与父类完全相同的方法头声明。

2）定义一个类

Java 中的每一个类都是从 Object 类继承而来。Object 类有两个常用方法：equal()方法和 toString()方法。equal()方法用于测试一个对象是否同另一个对象相等。toString()方法返回一个代表该对象的字符串，每一个类都会从 Object 类继承该方法，有些类重写了该方法，以便返回当前状态的正确表示。

定义一个类表示定义了一个功能模块。一个类的定义包含两部分内容，即类声明和类体。类是通过关键字 class 来定义的，在 class 关键字后面加上类的名称，这样就创建了一个类。说明部分还包括其继承的父类、实现的接口，以及修饰符 public、abstract 或 final。类体中定义了该类所有的变量和该类所支持的方法。

定义类的语法格式如下：

```
[修饰符] class 类的名称 [extends 父类的名称] [implements 接口的名称]{
  //类的成员变量
  //类的方法
}
```

下列代码定义了 racing_cycle 类，该类是一个公共类，描述的是一个公路赛车，其父类为 bicycle。

```
public class racing_cycle extends bicycle{
  //racing_cycle类的成员变量和方法
}
```

设计一个类要明确所要完成的功能，类里的成员变量和方法描述类的功能。成员变量就是这个类里定义的一些私有的变量，这些变量是属于这个类的。定义成员变量的语法如下：

```
变量的类型 变量的名称;
```

对类的成员可以设定访问权限，来限定其他对象对它的访问。访问权限有 private、protected、public 和 friendly 几种类型。

方法收到对象的信息后进行相关的处理。创建方法的语法如下：

```
[修饰符] 方法的返回类型 方法名称 ([参数列表]) {
    方法体
}
```

方法的返回值可以是任意类型,如 String、Boolean、int。如果定义了方法的返回类型就必须在方法体内用 return 语句把返回值返回。方法的返回值可以为 null,但必须是对象类型,在返回值为基本类型时,只要能够自动转换就可返回。

方法的参数可以是基本数据类型,也可以是对象引用类型。每个参数都要有完整的声明该变量的形式。方法的参数可以有一个,也可有多个。Java 程序的入口 main()就是一个方法,参数为 String[] args,它是个特殊的方法。

一个类的所有方法中有一个特殊的方法,称为"构造方法"。Java 中的每个类都有构造方法。构造方法用来初始化该类的一个新的对象。构造方法具有和类名相同的名称,而且不返回任何据类型。

3) 使用类创建对象

创建类的实例是通过 new 关键字来定义的,后面加上定义类时为类起的名称,需要注意的是在类名后还需要一个括号,括号中是构造方法的参数。创建类的实例的语法格式如下:

```
类名称 对象名称=new 类名称 (构造方法参数);
```

当用运算符 new 为一个对象分配内存时,会自动调用类的构造方法。用构造方法进行初始化避免了在生成对象后每次都要调用对象的初始化方法,而且构造方法只能由 new 运算符调用。由于对构造方法可以进行重载,所以通过给出不同个数或类型的参数会分别调用不同的构造方法。

用 new 运算符可以为一个类实例化多个不同的对象。这些对象分别占用不同的内存空间,因此改变其中一个对象的状态不会影响其他对象。

4) 引用对象的成员变量

对象引用就是该引用名称指向内存中的一个对象,通过调用该引用即可完成对该对象的操作。如果调用的对象或成员变量没有创建,那么在编译时编译器将出现空指针错误,因为成员变量和方法属于对象,即属于用 new 关键字创建出来的对象。通过 new 关键字来创建一个对象后,会有一个系统默认的初始值。所以不管有没有在创建成员变量的时候给变量一个值,系统都会有一个默认的值。

访问对象的某个成员变量,其格式如下:

```
objectReference.variable
```

其中 objectReference 是对象的一个引用,它可以是一个已生成的对象,也可以是能够生成对象的方法调用。

5) 调用对象的成员方法

调用对象的某个成员方法,其格式如下:

```
objectReference.method(Args)
```

1.3.8 异常处理

异常发生的原因有很多,可能是软件的问题,也可能是硬件的问题。在 Java 程序中,



一般通过 try-catch 语句来进行异常处理。try-catch 语句的基本语法如下：

```
try {
    //此处是可能出现异常的代码
}catch(Exception e) {
    //此处是发生异常时的处理代码
}[finally {
    //此处是无论是否发生异常都必须被执行的代码
}]
```

try 语句中是可能出现异常的代码；在 catch 子句中需要给出一个异常的类型和该类型的引用，并在 catch 子句中编写当出现该异常类型时需要执行的代码。

try-catch 语句是对有可能发生异常的程序进行检查，如果没有发生异常，就不会执行 catch 语句中的内容。在程序中如果不使用 try-catch 语句，则当程序发生异常的时候，会由系统处理，通常是自动退出程序的运行。而使用 try-catch 语句后，当程序发生异常的时候，会执行 catch 语句中的语句，从而使程序不自动退出。

try-catch 语句中的 catch 子句可能不只一个，可用多个 catch 子句定义可能发生的多个异常。当处理任何一个异常时，则将不再执行其他 catch 子句。所以当对程序使用多个 catch 语句进行异常处理时，特别需要注意的是要将范围相对小的异常放在前面，将范围相对大的异常放在后面。

在 try-catch 语句中还可以有 finally 子句，finally 子句中是无论是否发生异常都必须被执行的代码。在实际开发中经常要使用 finally 子句，例如，在数据库操作中，连接数据库时可能发生异常，也可能不发生异常，但是不管是否发生异常，连接数据库所用到的资源都是需要关闭的，这些操作是必须执行的，这些执行语句就可以放在 finally 子句中。

1.4　XML 基础

1.4.1　XML 简介

XML(eXtensible Markup Language)是一种可扩展的标记语言。标记语言是指在普通文本中加入一些具有特定含义的标记(Tag)，以对文本的内容进行标识和说明的一种文件表示方法。

作为一种标记语言，XML 与 HTML 类似，但并非 HTML 的替代。XML 和 HTML 是为不同目的而设计的，HTML 被设计用来显示数据，其重点是数据的外观，而 XML 被设计用来传输和存储数据，是一种独立于软件和硬件的信息传输工具，其重点是数据的内容。

作为一种标记语言，XML 最基本、最主要的功能就是在文档中添加标签。如下例所示，在＜＞和＜/＞里面的文本，就是一些“标签”。标签必须成对出现，如＜book＞和＜/book＞、＜name＞和＜/name＞、＜country＞和＜/country＞等。下面是一段示例代码。

```
<?xml version="1.0" standalone="yes"?>
<book>
    <name>Android Programming Guide</name>
    <author>
        <name>Zhang</name>
        <sex>male</sex>
        <age>45</age>
        <country>China</country>
    </author>
    <price>35.5</peice>
</book>
```

在 HTML 文档中只使用在 HTML 标准中定义过的标签,如<p>、<h1> 等。与此不同的是,XML 所使用的标记都是非预定义的,被设计为具有自我描述性。XML 允许作者定义自己的标签和自己的文档结构,只要遵守 XML 的标记命名规则,可以在文档中添加任何标记。如在上例中,可以将<book>、</book>标签改为<BookInformation>和</BookInformation>,也可以改为<BK>和</BK>。用户可以自定义标记,这就是 XML 称为"可扩展"标记语言的由来。

XML 没有任何行为。XML 被设计用来结构化、存储以及传输信息,其本身仅仅是纯文本。如上例所示的 XML 文档,文档中有书名、作者等信息。但是,这个 XML 文档并没有做任何事情,它既不能像程序一样运行,也不会有任何运行结果。它仅仅是包装在 XML 标签中的纯粹的信息,同样也不描述其如何显示、输出等格式化信息。若要格式化文档的输出,需要另外编写控制其输出的样式表文件。若要传送、接收和显示出这个文档,也需要另外编写软件或者程序。

对于自定义的标记,用户可在文档内,也可以在文档之外进行说明。当然也可以不进行说明。XML 对所使用的标记进行说明的部分,称为 DTD(Document Type Definition),即文档类型定义。DTD 定义了用户所使用的所有标记以及标记之间的逻辑关系,同时也定义了文档的逻辑结构。一个 XML 文档若包含了 DTD,应用程序就可以根据 DTD 的定义来检查文档的完整性和正确性。

在浏览器中可查看 XML 文件,但是由于 XML 文档本身不会携带有关如何显示数据的信息,其标签是由 XML 文档的作者创建的,浏览器无法确定文档中标签的具体含义,所以大多数浏览器仅把 XML 文档显示为源代码。如上例所描述的 XML 文档,它在 IE 中的显示结果如图 1-9 所示。XML 文档将显示为代码颜色化的根以及子元素。通过单击元素左侧的加号或减号,可以展开或收起元素的结构。如需查看源代码,可以从浏览器菜单中执行"查看"→"源文件"命令。如果浏览器打开了某个有错误的 XML 文件,那么它会报告这个错误。

虽然浏览器对文档进行了语法分析,文档内容、指令和标记分别被显示成不同的颜色,但一般来讲,我们需要显示的只是文档的原始内容,指令和标记作为附加的信息在实际显示时应该被隐藏起来,并且书名和作者信息等不同级别的信息要使用不同的字体和字号。要达到这个目的,就要为文档编写样式表。在 XML 中,内容和显示是分离的,标

记的显示方案在 XML 文档中附带的样式文件中定义。这也是 XML 与 HTML 之间的一个重大差别。

控制 XML 文档的显示格式，可以使用 CSS、XSLT 和 JavaScript 等方法。其中使用 XSLT（eXtensible Stylesheet Language Transformations）显示 XML 是首选。使用 XSLT 的方法有两种模式，一种是在浏览器显示 XML 文件之前，先把它转换为 HTML，另一种模式是在服务器上进行 XSLT 转换。前者 XSLT 转换是由浏览器完成的，不同的浏览器可能会产生不同结果。在 Android 编程的过程中很少用到此部分内容，所以本书不做详细介绍，有兴趣的读者请参阅相关文献。

内容和显示分离，不仅提高了输出形式的灵活性，还具有更高的弹性。文件组织者可以不再考虑文件的输出格式，甚至都可以不考虑文件的用途，而只需要尽可能完美地描述文件的内容。一个 XML 文档，可以被有着各种不同目的的用户进行各种各样的处理。不同用户可以使用其不同的部分，可以用来显示，也可以用来打印，或者被输入到数据库……大家各取所需，各尽其用。因此 XML 比 HTML 具有更高的弹性和灵活性。



图 1-9 XML 文档在 IE 中的显示结果

1.4.2 XML 的用途

XML 的主要用途是各种应用程序之间进行数据传输，它在信息存储和描述领域变得越来越流行。

XML 可以简化数据共享。各个计算机系统和数据处理平台使用不兼容的格式来存储数据，而 XML 数据以纯文本格式进行存储，因此提供了一种独立于软件和硬件的数据存储方法。这让创建不同应用程序可以共享的数据变得更加容易。

XML 可以简化数据传输。通过 XML，可以在不兼容的系统之间轻松地交换数据。对开发人员来说，在因特网上的不兼容系统之间交换数据是一项非常费时费力的工作。由于可以通过各种不兼容的应用程序来读取数据，以 XML 交换数据可以使这一类工作更简单。

XML 可以简化平台的变更。升级到新的系统，无论是升级硬件还是升级软件，总是

非常费时的。必须转换大量的数据，不兼容的数据经常会丢失。XML 数据以文本格式存储。这使得 XML 在不损失数据的情况下，更容易扩展或升级到新的操作系统、新应用程序或新的浏览器。

XML 可以使数据更有用。由于 XML 独立于硬件、软件以及应用程序，XML 使数据更可用，也更有用。不同的应用程序都能够访问 HTML 网页或 XML 数据源中的 XML 数据。另外，通过 XML，数据还可以供计算机、语音设备、新闻阅读器等各种设备使用。

1.4.3 XML 文档的结构

XML 使用了简单的具有自我描述性的语法，采用一种有逻辑的树型结构。XML 文档必须包含根元素，该元素是所有其他元素的父元素；文档中的元素形成了一棵文档树，这棵树从根部开始，并扩展到树的叶端；所有元素均可拥有子元素；相同层级上的子元素为兄弟元素；所有元素均可拥有文本内容和属性。

以下是一个 XML 文档的示例。

```
<?xml version="1.0" encoding="gb2312" standalone="yes"?>
<computerbooks>
    <book>
        <bookname>Android Programming Guide</name>
        <author>
           <name>Zhang</name>
           <country>China</country>
        </author>
        <price>35.5</price>
    </book>
    <book>
        <bookname>XML Tutorial </name>
        <author>
           <name>Mark</name>
           <country>Canada</country>
        </author>
        <price>38</price>
    </book>
</computerbooks>
```

文档中的第一行是 XML 声明，它定义了 XML 的版本和所使用的编码。

第二行描述文档的根元素＜computerbooks＞，文档中的所有＜book＞元素都是根的子元素，都被包含在＜computerbooks＞中。每个＜book＞元素还有 3 个子元素＜bookname＞、＜author＞、＜price＞，最后一行定义根元素的结尾＜/computerbooks＞。整个文档的逻辑结构如图 1-10 所示。

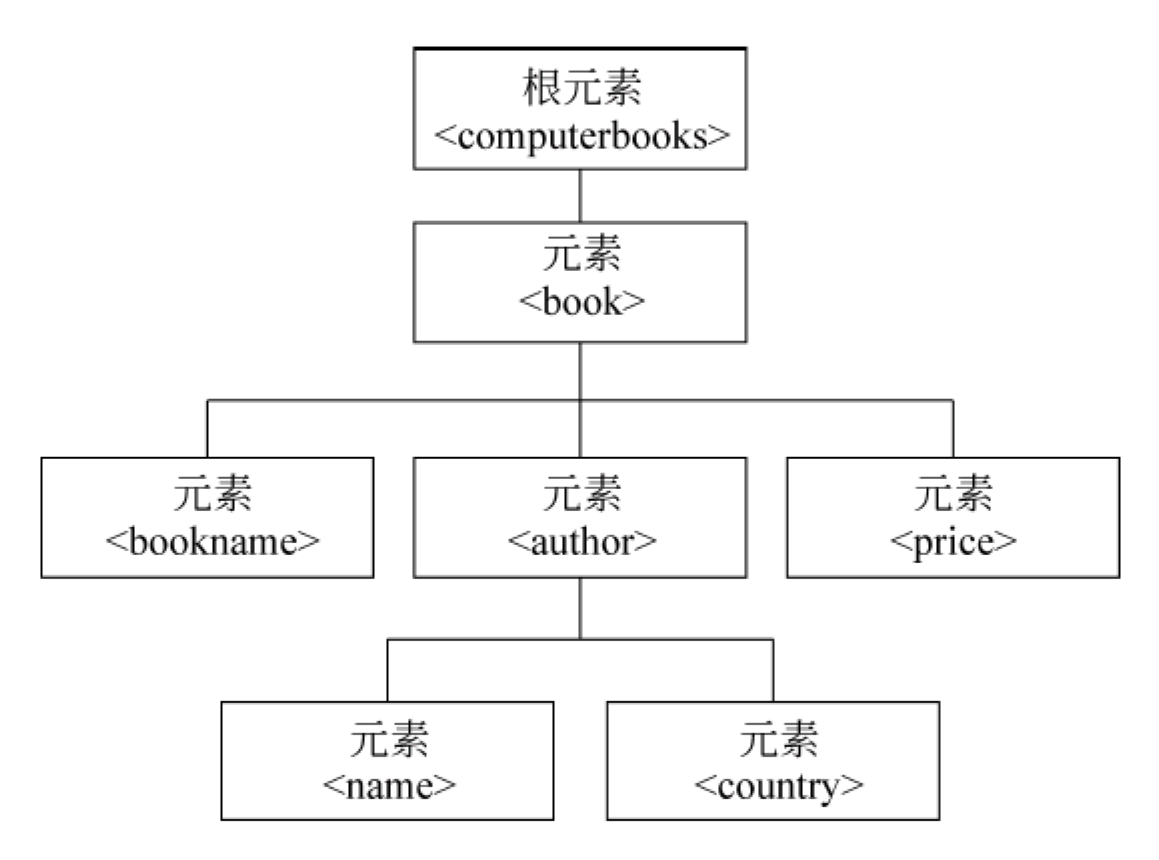


图 1-10　XML 文档的逻辑结构

1.4.4　XML 语法

1．声明部分

XML 文档的声明(Declaration)部分又称为“前言”(Prolog)。XML 声明是一条 XML 指令,位于文档的首行。例如:

```
<?xml version="1.0" encoding="gb2312" standalone="yes"?>
```

该行的内容包括如下内容。

(1) <? …? >:表示该行是一条指令。

(2) xml:表示该文件是一个 XML 文件。

(3) version="1.0":表示该文件遵循的是 XML 1.0 标准。

(4) encoding="gb2312":表示该文件使用的是 gb2312 字符集。

(5) standalone="yes":表示该文件未引用其他外部的 XML 文件。

XML 声明必须是文档的首行,且必须从第一个字符开始,前面不能有包括空格在内的任何其他字符。因为即使是简单的英文字符串,也可能有不同的编码方式,在开始分析文档声明的时候,解析器并不知道文档使用了何种字符集。此时解析器就要读取文档最前面的几个字符,与字符串“<? xml”的不同字符集下的编码进行比较,以确定文档所使用的编码方式。确定了编码方式后,才能够做进一步的读取和分析工作。如果文档声明前有其他字符,解析器取出的前几个字符并不是“<? xml”,无法与标准的“<? xml”字符串进行比较,解析就会失败。

XML 指令与标签一样,都不属于文档的内容,都是根据 XML 规范添加进文档的附加信息。但标签用于标注文档的内容,而指令则用于控制文档。无论是解析器还是最终处理 XML 文档的应用程序,都要根据指令所提供的控制信息对文档进行分析,否则,将无法正确解读文档。

文档声明行在 XML 文档中非常重要,几乎所有的 XML 文档都要有,当然其具体内

容可能有差别。只有当文档所使用的字符集,即 encoding 属性的值为 UTF-8 或 UTF-16,而且文件未引用其他外部的 XML 文件,即 standalone 属性的值为 yes 时,才可以省略这一行。但 XML 1.0 标准强烈建议无论何种情况都保留文档声明行,且位于文档的第一行。

2. XML 元素

XML 元素使用 XML 标签进行定义。XML 的语法规则要求所有 XML 元素都必须有开始标签和结束标签。元素可包含其他元素、文本或者两者的混合物。标签实际上就是标记的名称,同时也是元素名。元素的名称可以含字母、数字以及其他的字符,不能以数字或者标点符号开始,不能以字符 xml(或者 XML、Xml)开始,名称不能包含空格。

为了更准确清晰地反映文档的内容,元素名称应具有描述性,避免使用类似"-"、"."、":"这样的字符,因为有些软件认为这些字符有特殊的含义,会引起文档内容的误读。在 XML 中,与简洁性相比更重要的是准确和清晰,这是 XML 的原则之一,这与编程中变量的命名原则是相似的。

XML 标签对大小写敏感,必须使用相同的大小写来编写开始标签和结束标签。元素也可以拥有属性,也可以包含其他元素,这就构成了元素的嵌套。对于元素的嵌套,有如下原则。

(1) 所有 XML 文档都从一个根节点开始,该根节点代表文档本身,根节点包含了一个根元素。

(2) 文档内所有其他元素都包含在根元素中。

(3) 包含在根元素中的第一个元素称为根元素的子元素。如果不止一个子元素,且子元素没有嵌套在第一个子元素内,则这些子元素互为兄弟。

(4) 子元素还可以包含子元素。

所有元素都必须彼此正确地嵌套。元素进行嵌套时,必须注意不能交叉、重叠。一个元素 A 如果含有子元素 B,则子元素 B 的开始标签和结束标签都必须位于元素 A 之内,不能一个在 A 内,一个在 A 外。

例如,如下代码是正确的:

```
┌<book>
│   ┌<author>
│   │    <name>Zhang</name>
│   │    <country>China</country>
│   └</author>
└</book>
```

但下例中的元素嵌套就不正确:

```
┌<book>
│   ┌<author>
│   │    <name>Zhang</name>
│   │    <country>China</country>
└</book>
    └</author>
```

3. XML 属性

类似于 HTML，XML 元素可以在开始标签中包含属性(Attribute)，XML 属性提供关于元素的额外信息。属性通常提供不属于数据组成部分的信息，但是对需要处理这个元素的软件来说却很重要。

属性由＝分隔开的名称-数值对构成，格式如下：

```
<元素名　属性名="属性值"…>内容</元素名>
```

或：

```
<元素名　属性名="属性值"…/>
```

例如：

```
<Price MoneyKind="RMB">22000</Price>
<Rectangle Width="100" Height="80"/>
```

属性值必须被引号包围，单引号和双引号均可使用。如果属性值本身包含双引号，那么可以使用单引号包围它，例如：

```
<author name='Jangle "MM" Smith'>
```

或者可以使用实体引用：

```
<author name="Jangle &quot;MM&quot; Smith ">
```

应尽量使用元素来描述数据，而仅仅使用属性来描述附加性信息或与数据无关的信息。因为使用属性可能会引起一些问题，例如，属性无法包含多个值，属性无法描述树结构，难以阅读和维护等。

至于什么样的信息是元素或内容的“附加性”信息，并没有一个明确的规定，一般来讲，与文档的内容无关的无子结构信息，例如元素＜MyDocument LastUpdate＝"2014/10/19"＞…＜/MyDocument＞中，更新时间与内容无关，可以考虑使用属性进行描述。

通常，在将已有文档作为 XML 文档处理时，文档的原始内容应全部表示为元素。编写者所增加的一些附加信息，如对于文档某一点内容的简单说明、注释等，可以表示为属性。另外，希望读者看到的内容应表示为元素，反之表示为属性。

4. 实体引用

非法的 XML 字符必须被替换为实体引用(entity reference)，这类似于编程语言中的“转义字符”。例如，在 XML 文档中元素内容的位置出现一个＜字符，这个文档会产生一个错误，这是因为解析器会把它解释为新元素的开始。为了避免此类错误，需要把字符＜替换为实体引用。

错误的写法：

```
<message>if n<10 then</message>
```

正确的写法:

```
<message>if n &lt; 10 then</message>
```

在 XML 中有 5 个预定义的实体引用,分别是"<(<)"、">(>)"、"&(&)"、"'(')"、""(")"。

5. 注释

注释用于对语句进行某些提示或说明。解析器分析文档时,将完全忽略注释中的内容。

XML 文档的注释起始和终止界定符分别为"<!——"和"——>"。注释有如下规则。

(1) 注释不能出现在 XML 声明之前。

XML 声明必须是文档的首行。例如,下面的文档是非法的:

```
<!--This is my first xml document-->
<?xml version="1.0" standalone="yes"?>
<bookName>
    Android Programming Guide
</bookName>
```

(2) 注释不可以出现在标记中。

下面的注释是非法的:

```
<bookName <!--This is my first xml document-->>。
```

(3) 注释中不可以出现连续两个连字符,即——。

下面的注释是非法的:

```
<!--This is--my first xml document-->
```

(4) 注释中可以包含元素,但是元素中不能包含——字符。

这时此元素也成为注释的一部分,在解析时将被忽略。例如,下面的注释是合法的:

```
<!--This is my first xml document
<bookName>Android Programming Guide</bookName>
End!-->。
```

(5) 注释中的关键字符,如小于号(<)、大于号(>)、单引号(')、双引号(")、与字符(&),都需要使用预定义实体引用进行代替。

例如,某一注释的内容为"This's a "my" document",则该注释的正确写法如下:

```
<!--This&apos;s a &quot;my&quot; document-->。
```

1.4.5 XML 命名空间

使用 XML 可以创建不同的标记语言,可以将使用不同标记语言创建的文档组合使

用。但是，在这些不同的标记语言下，可能定义了一些意义不同而名称相同的标记。这时候将两种文档混合，这些同名的标记将导致混乱。命名空间可以解决这个问题。

命名空间通过在元素名前增加一个独特的标识符，来标识元素的活动领域，这个标识符必须是独一无二的。XML 使用因特网上的网址作为标识符，因为因特网上的网址肯定是独一无二的。但网址中含有 XML 标识符中禁止使用的字符，如每个网址都要使用的/；另外网址一般都很长，在文档中的许多元素名前都增加一个很长的前缀，输入和阅读都不方便。所以，XML 采取了使用“前置字串”(prefix)的方法，即把用来作为标识符的网址定义为一个前置字串，在文档中使用这个前置字串代替网址，对元素名进行标示。

XML 文档中定义命名空间的语法如下：

```
<element_name  xmlns:prefix="URI">
```

或

```
<prefix:element_name  xmlns:prefix="URI">
```

需要说明的是，作为标识符的网址，在命名空间中只是起一个标识作用，而并不是真的要使用该网址下的文档或者规则。所以该网址的精确性并不重要，它甚至可以根本就不存在。

命名空间具有作用范围。这个范围是指 XML 文档树状结构中的层次关系。父元素定义的命名空间可以用在子元素上，即父元素定义的命名空间的作用范围包含子元素。但子元素中定义的命名空间不可以包含父元素。所以，命名空间一般在根元素中定义。如果一定要在文档中间定义命名空间，则一定要确保所有属于该命名空间的元素都包含在定义时所确定的作用范围之内。

例如，以下 XML 文档描述某个表格中的信息：

```
<table>
    <tr>
        <td>Coffee</td>
        <td>Tea</td>
    </tr>
</table>
```

另一个 XML 文档描述有关桌子的信息：

```
<table>
    <name>Coffee Table</name>
    <width>80</width>
    <length>120</length>
</table>
```

如果这两个 XML 文档一起使用，由于两个文档都包含带有不同内容和定义的<table>元素，就会发生命名冲突。使用命名空间可以避免冲突。

使用了命名空间的 XML 文档：

```
<h:table xmlns:h="http://hebusta.edu.cn/table">
    <h:tr>
        <h:td>Coffee</h:td>
        <h:td>Tea</h:td>
    </h:tr>
</h:table>
<f:table xmlns:f="http://hebustb.edu.cn/furniture">
    <f:name>Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

可以像上例一样,将 XML 命名空间属性放置于某个元素的开始标签之中。当一个命名空间被定义在某个元素的开始标签中时,所有带有相同前缀的子元素都会与同一个命名空间相关联。

1.5 本 章 小 结

本章首先学习智能移动设备的概念、常见智能移动设备操作系统以及 Android 系统的体系结构及其优点,然后学习了 Android 程序设计必要的预备知识,包括 Java 语言基础和 XML 的相关知识等。掌握本章的知识可以为以后的学习打下基础。

习 题

1. Android 操作系统与其他常见的智能移动设备操作系统相比有哪些优点?
2. 简述 Android 系统的体系结构。
3. Android 的 Dalvik 虚拟机有什么优点?
4. 简述 Java 标识符的定义规则,指出下面的标识符中哪些是不正确的,并说明理由。

Here,_there,this,it,2to1,_it,a_123,boolean,$abc,name,myAge

5. 编写显示下列图形的 Java 程序:

```
*******
 *****
  ***
   *
```

6. 编写一个 Java 应用程序,以字符串的格式输出当前的日期和时间。
7. 编写一个 Java 应用程序,利用数组打印连续小写字母。
8. 编写一个 Java 应用程序,实现以下功能:若原工资大于等于 3000 元,增加工资 10%;若原工资小于 3000 元大于等于 2000 元,则增加工资 15%;若原工资小于 2000 元则增加工资 20%。请根据用户输入的工资,计算出增加后的工资并显示输出。

9. 阅读下列程序段，写出其输出结果。

```
void complicatedexpression_r() {
    int x=20, y=30;
    boolean b;
    b=x>50&&y>60||x>50&&y<-60||x<-50&&y>60||x<-50&&y<-60;
    System.out.println(b);
}
```

10. 阅读下列程序段，写出其输出结果。

```
public class Exercises1_10 {
    public static void main(String[] args) {
        int n=1, m, j, i;
        for(i=3; i<=30; i+=2) {
            m=(int)Math.sqrt((double)i);
            for(j=2; j<=m; j++)
                if((i%j)==0)
                    break;
            if(j>=m+1) {
                System.out.print(i+"   ");
                if(n%5==0)
                    System.out.print("\n");
                n++;
            }
        }
    }
}
```

11. 阅读下列程序段，写出其输出结果。

```
class Aclass {
    void go() {
        System.out.println("Aclass");
    }
}
public class Bclass extends Aclass {
    void go() {
        System.out.println("Bclass");
    }
    public static void main(String args[]) {
        Aclass a=new Aclass();
        Aclass b=new Bclass();
        a.go();
        b.go();
    }
```

```
}
```

12. 什么是对象？什么是类？二者有何关系？
13. 简述构造方法的特点与作用。
14. 什么是继承？继承的特性可给面向对象编程带来什么好处？
15. 了解 XML 技术，简述 XML 文档的组成成分及其作用。
16. 编写一个 XML 文档，描述计算机系课程的设置及相关信息。
17. 试分析以下 AndroidManifest.xml 文档的格式和语句内容。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hebust.zxm.broadcastreceiverexample"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="edu.hebust.zxm.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

第2章 创建第一个Android应用程序

本章首先介绍在Windows系统中搭建Android应用程序开发平台的主要步骤以及集成开发环境的使用方法；然后通过学习创建第一个Android应用程序，进一步熟悉Android集成开发环境，了解典型Android应用程序的架构与组成。

2.1 搭建Android应用程序开发环境

开发Android应用程序，可以在Windows、Linux等平台上完成。本书以Windows平台为例，介绍Android应用程序开发环境的搭建过程，在其他系统平台上搭建开发环境的方法与此类似，可参阅相关文献。

搭建Android应用程序开发环境需要JDK、Eclipse、ADT插件和Android SDK等。

2.1.1 集成开发环境的下载与安装

Google公司选择Eclipse作为其开发Android应用程序的集成开发环境，并开发了Eclipse外挂工具ADT(Android Development Tools)。ADT在常规的Eclipse中创建了一个Android专属的开发环境，并扩展了Eclipse的功能，可以让程序开发者快速、方便地建立和调试Android工程项目，包括创建Android应用程序、将其发送到Android仿真环境中运行和测试、在基于Android框架的API上添加组件，以及用SDK工具集(如DDMS等)调试应用程序、导出签名的APK程序以便发布应用程序等功能。

Android SDK提供的API是一个庞大的程序设计实例库，它包含应用程序、属性提供、图形处理、多媒体、操作系统、文字编辑与显示组件等工具。可以到网址http://developer.android.com/sdk/index.html，根据自己的操作系统选择下载合适的Android SDK版本。

Android提供了一个集成Eclipse、ADT插件、Android SDK和Android平台工具的SDK版本，如图2-1所示。下载文件名为adt-bundle-windows-x86-20140321.zip的文件，其后缀的数字是版本号，与下载时间有关。本书以此版本为例，介绍环境的搭建和应用程序的开发。

在运行开发环境之前，需要首先安装和配置JDK。JDK的下载、安装和配置过程见第1章，不再赘述。



图 2-1 下载 Android 的 SDK

下载完 Android 的 SDK 文件后，把该文件解压到计算机上某个位置上即可。解压后的文件夹中包括文件夹 eclipse、sdk 和文件 SDK Manager. exe，如图 2-2 所示。

Android 集成开发环境使用的平台是 Eclipse，所以要启动开发环境，直接到解压文件夹中的 eclipse 文件夹中找到文件 eclipse. exe，运行此文件即可。Android 开发环境的启动画面如图 2-3 所示。



图 2-2 解压后的文件夹

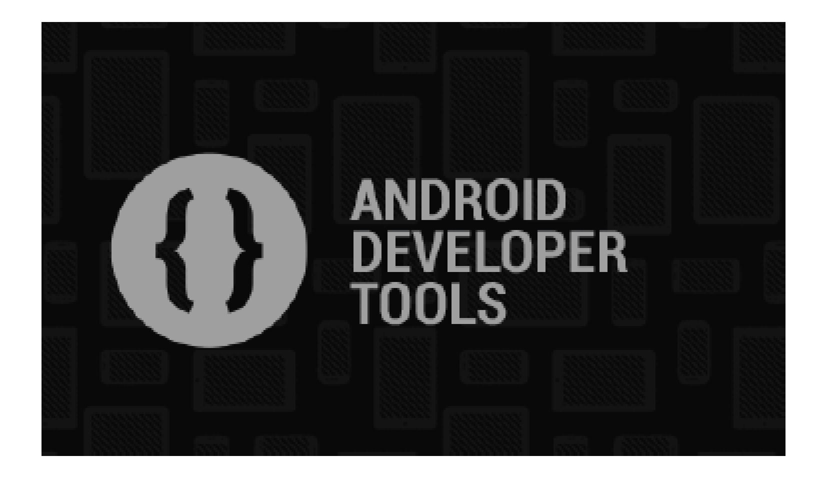


图 2-3 集成开发环境的启动画面

2.1.2 开发环境简介

在使用 Eclipse 环境之前，需要设置 SDK 的路径。在 Eclipse 菜单中选择 Window→Preferences。打开参数设置对话框，如图 2-4 所示。在左侧列表中选择 Android 项，然后在右侧 SDK Location 栏中设置 SDK 的路径为开发环境解压路径下的 SDK 文件夹。

设置完成后，就可以创建 Android 工程项目了。Eclipse 开发环境的界面如图 2-5 所示。

Eclipse 开发环境窗口中包含菜单栏、工具栏、状态栏和各种面板，图 2-5 中所示为

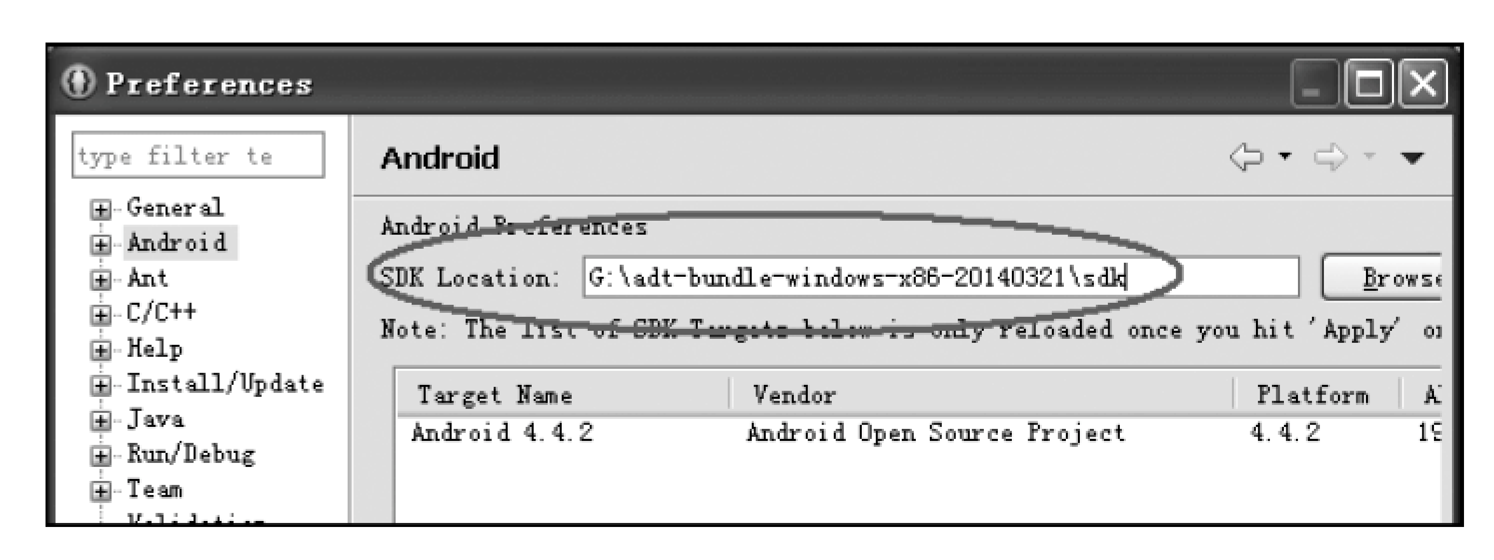


图 2-4　设置 SDK

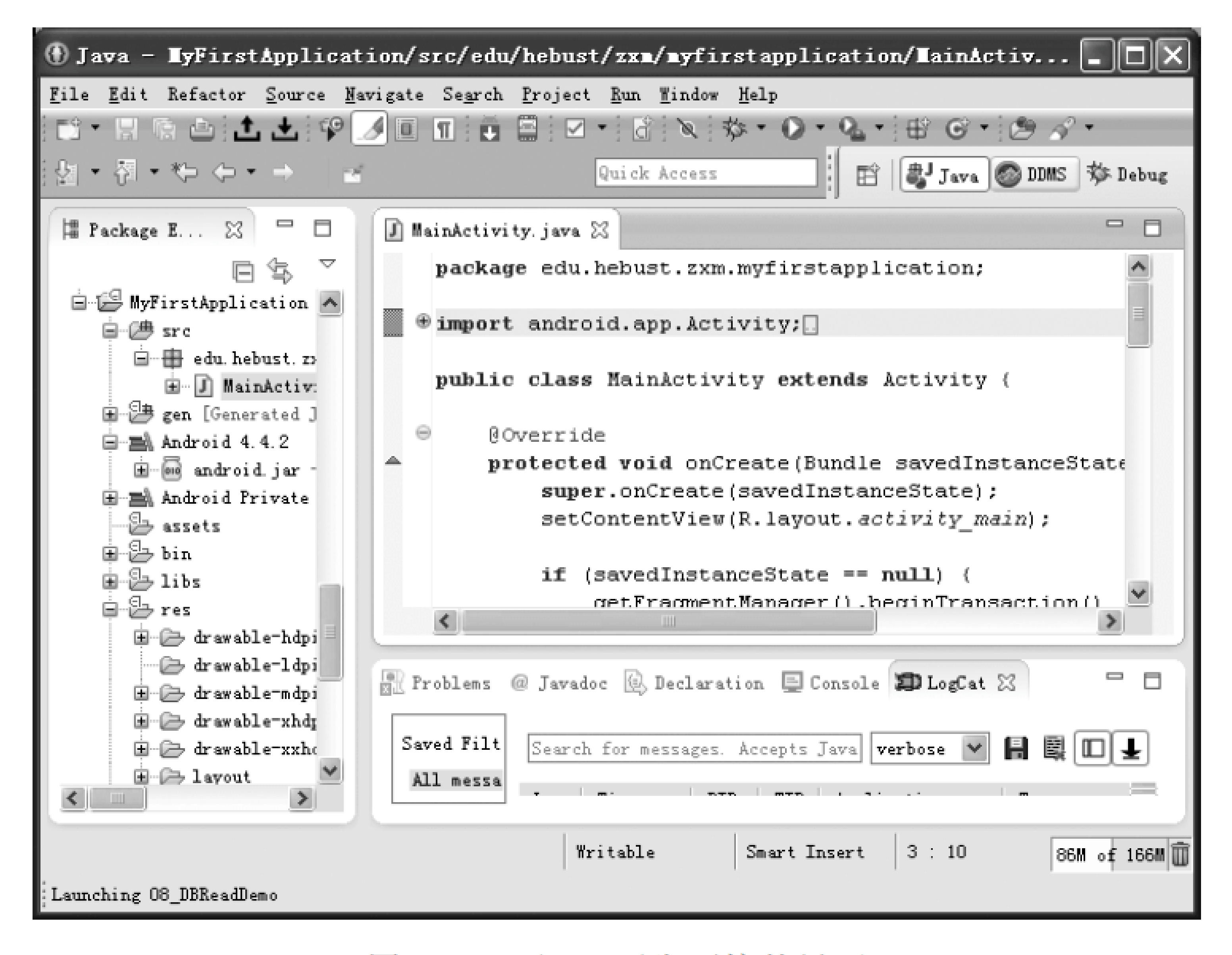


图 2-5　Eclipse 开发环境的界面

Package Explorer 面板和源代码面板，这是编辑应用程序最常用的两个面板。Package Explorer 面板中显示所有工程项目的树型目录结构，在这个目录结构中找到某个文件双击，就可以在右侧的源代码面板中显示其文件内容，并可以对其编辑修改。

在 Eclipse 窗口中，执行 File→New→Android Application Project 命令，或在工具栏中执行 New→Android Application Project 命令，可以创建一个新的 Android 应用程序。

执行 File→Import→Android→Exiting Android Code Into Workspace 命令，可以将一个已有的 Android 项目文件导入到当前工作空间。

在左侧 Package Explorer 面板中选中某个工程项目，执行 Run→Run As→Android Application 命令，或直接右击，依次选择 Run As→Android Application，则编译运行该项目。

2.1.3　创建和启动 Android 虚拟设备 AVD

设置了 Java JDK、Eclipse 集成开发环境、Android SDK 路径后，就可以开始编写

Android 应用程序了。在 Eclipse 中完成应用程序的开发后，可以先在虚拟设备上仿真运行而不必将其真正放到手机上运行。

虚拟设备 AVD(Android Virtual Device)可以帮助程序开发人员在计算机上模拟真实的移动设备环境来测试所开发的 Android 应用程序。在 Android 程序开发过程中，需要创建至少一个 AVD，每个 AVD 模拟了一套设备来运行 Android 平台。

既可以在 Eclipse 环境中创建 AVD，也可以在命令行中创建 AVD。本节介绍如何在 Eclipse 中创建 AVD，在命令行中创建的方法请读者参阅相关文献。

在 Eclipse 菜单中选择 Window→Android Virtual Device Manager，如图 2-6 所示。

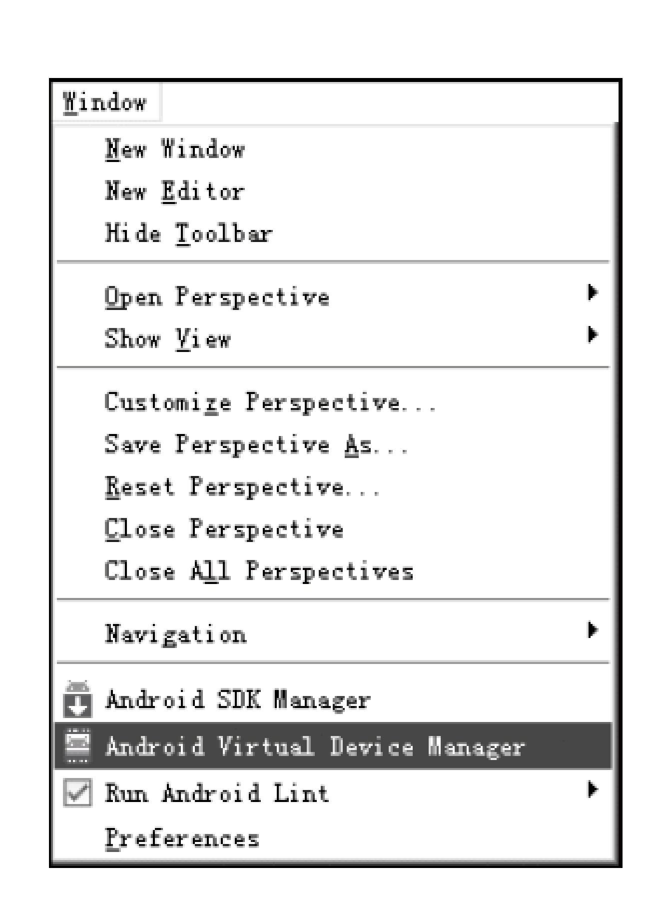


图 2-6　Window 菜单

打开 Android 虚拟设备管理器对话框，如图 2-7 所示。

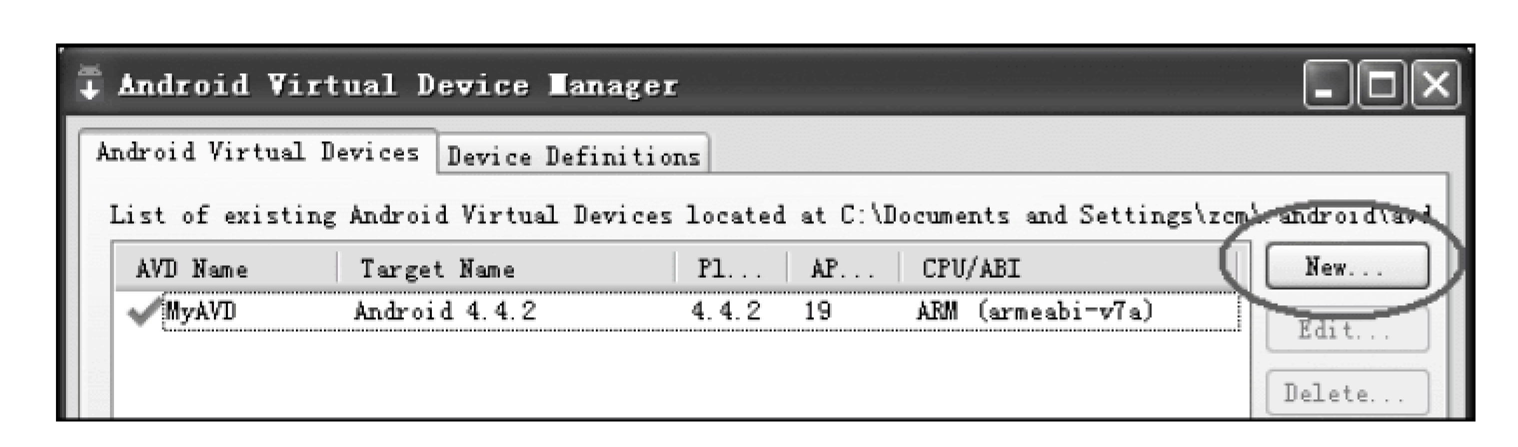


图 2-7　Android 虚拟设备管理器对话框

在对话框中单击 New 按钮，弹出如图 2-8 所示的对话框。在其中设置要创建的 AVD 的名称、Android API Leve、AVD 皮肤、SD 卡的大小等，单击 OK 按钮完成 AVD 创建。

创建完成之后，会在图 2-7 中列出所有已经创建的 AVD 模拟器。选中某个 AVD 模拟器，单击 Start 按钮，可以启动它。在 AVD 模拟器启动后，手机画面是上锁的，向右拖动画面中的锁图标，就可以解锁。启动后的模拟器如图 2-9 所示。

模拟器第一次启动时是英文界面，单击右侧的 MENU 按钮，选择 System settings 选项，弹出如图 2-10(a)所示的系统设置界面，单击其中的 Language&input，进一步选择 Language→"中文(简体)"选项，就可以将模拟器的界面设置为中文。

模拟器支持中文输入。如果在模拟器中不能输入中文，可单击如图 2-10(b)所示的"语言和输入法"选项，在弹出的界面中勾选"谷歌拼音输入法"即可。

在 Eclipse 中运行 Android 应用程序时，如果模拟器处于关闭状态，ADT 会自动启动默认的模拟器，并在其中运行。模拟器的启动比较耗时，所以在启动之后不要再关闭，每次运行应用程序时 ADT 会自动使用这个已经启动的模拟器，这样比较节省时间。

在 Eclipse 中运行开发的 Android 应用程序时，也可以选择在某个指定的 AVD 而不是默认的 AVD 上运行。方法是在 Eclipse 中右击某个工程，在弹出的快捷菜单中执行 Run As→Run Configurations 命令，在弹出对话框的 Target 标签下，选择在哪个 AVD

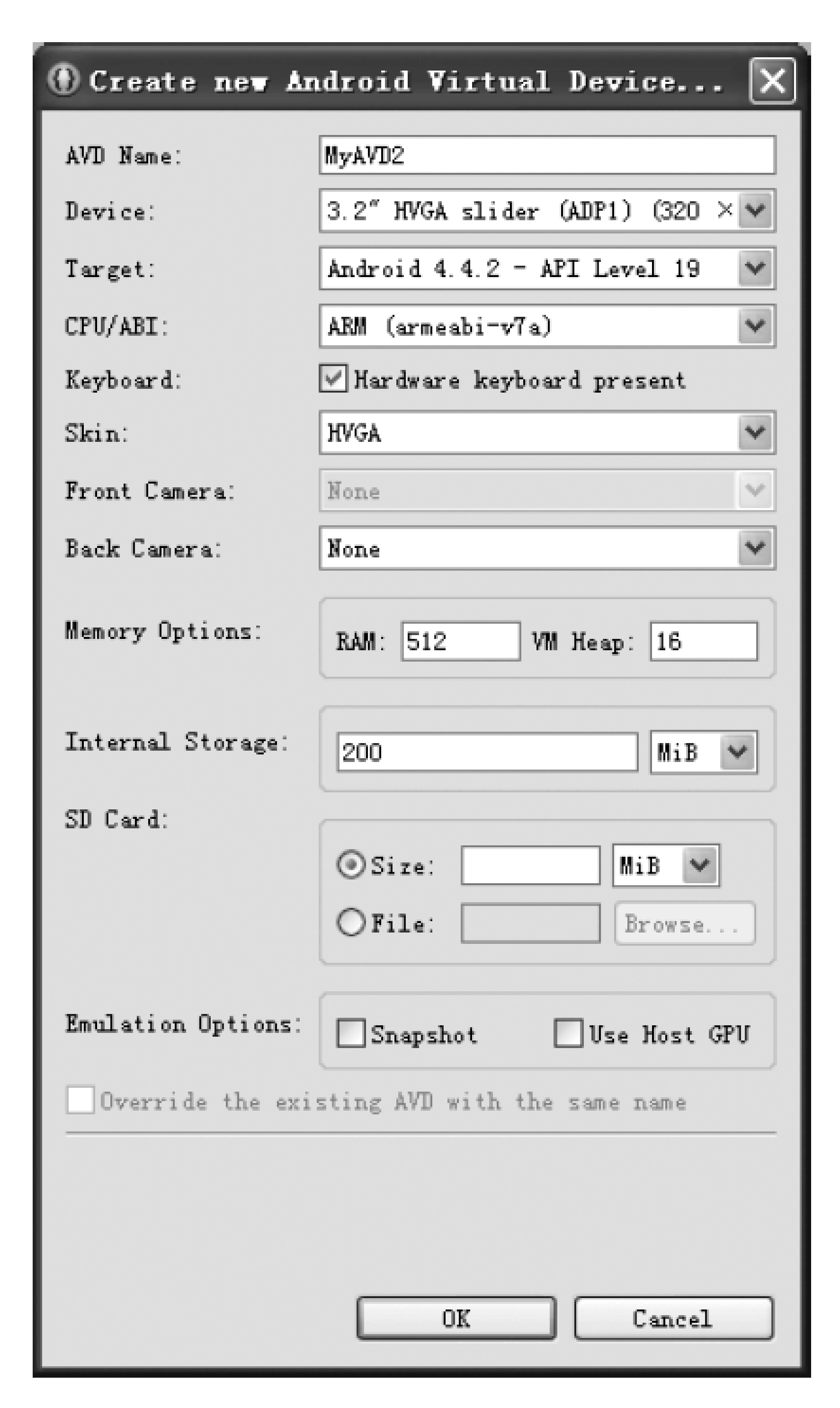


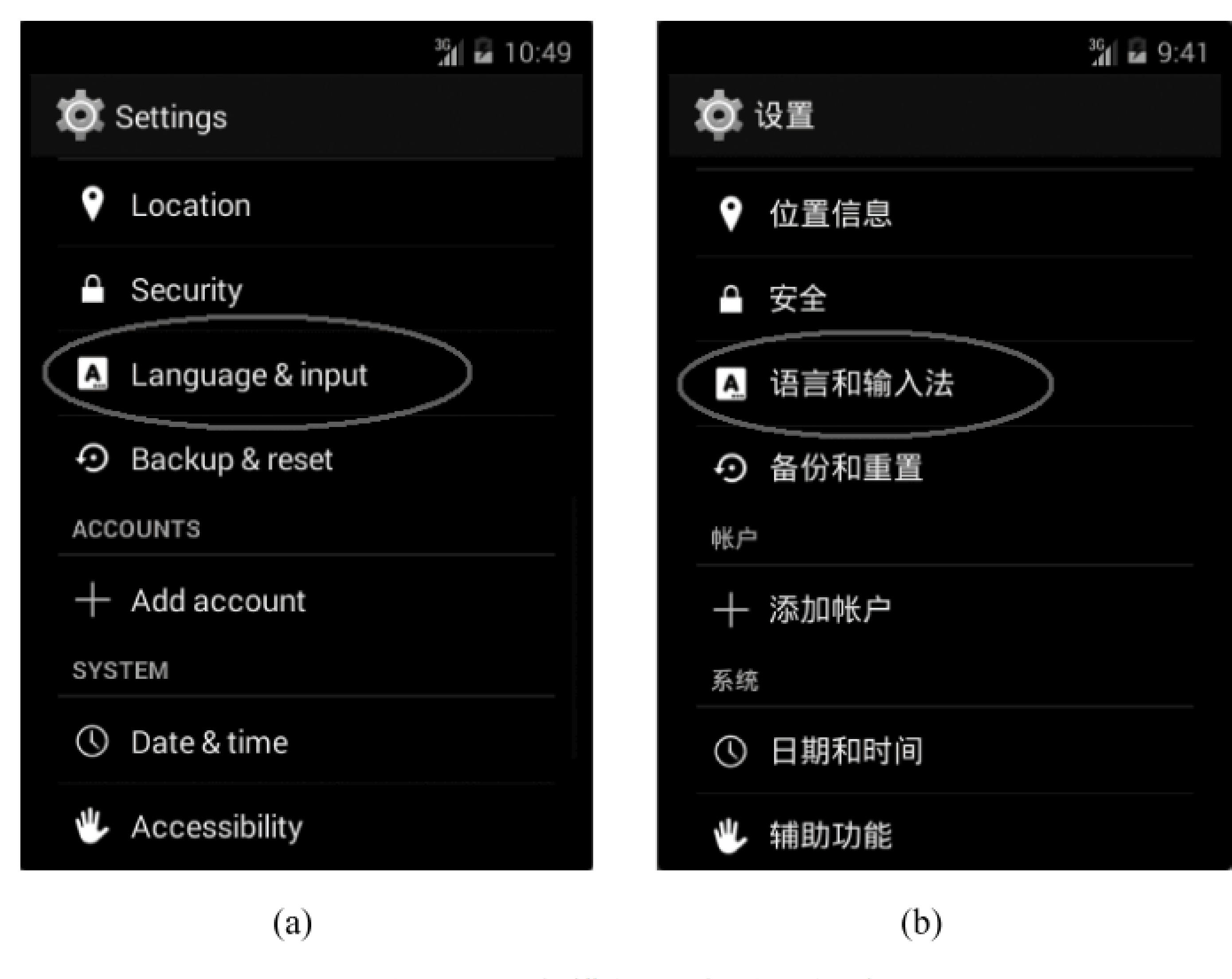
图 2-8　创建 Android 虚拟设备对话框



图 2-9　Android 模拟器

上运行应用程序。

需要注意的是,模拟器 AVD 毕竟不是真实的手机,有一些真实手机的功能模拟器是不能实现的,如不支持实际呼叫和接听电话、不支持 USB 连接、不支持照片和视频的捕获、不能确定电池水平和充电状态、不能确定 SD 卡的插拔等。



(a) (b)

图 2-10 在模拟器中设置语言

2.2 创建第一个 Android 应用程序的过程

在 Android 应用程序开发环境已经搭建起来之后,就可以创建 Android 应用程序。Android SDK 工具使用一套默认的项目目录和文件,能够很容易地创建一个新的 Android 工程项目。

221 新建 Android 工程项目

步骤 1:启动 Eclipse。

步骤 2:在 Eclipse 窗口中,执行 File→New→Android Application Project 命令,或工具栏中的 New→Android Application Project 命令,会显示如图 2-11 所示的对话框,开始建立应用程序。

步骤 3:在对话框中填写以下内容。

Application Name:显示给用户的应用程序名称,一般与工程名相同,本例使用 MyFirstApplication。当安装该应用程序到模拟器上后,在模拟器中的应用程序列表中就会看到这个名称。当应用程序在模拟器上运行时,该名称将显示在应用程序的标题栏。

Project Name:工程目录和 Eclipse 中显示的名称,物理上就是该项目对应文件夹的名字。

New Android Application

New Android Application
Creates a new Android Application

Application Name: MyFirstApplication
Project Name: MyFirstApplication
Package Name: edu.hebust.zxm.myfirstapplication

Minimum Required SDK: API 14: Android 4.0 (IceCreamSandwich)
Target SDK: API 19: Android 4.4 (KitKat)
Compile With: API 19: Android 4.4 (KitKat)
Theme: None

The application name is shown in the Play Store, as well as in the Manage A
Settings.

< Back　Next >　Finish　Cancel

图 2-11　新建工程对话框(一)

Package Name：应用程序的包命名空间。它使用与 Java 语言相同的包规则。在 Android 系统上安装的所有包中，自己的包名必须是唯一的。Sun 公司推荐的避免包名冲突方法是把开发组织的域名倒过来写，例如，google 的包名是 com.google。本书示例工程的包名统一采用 edu.hebust.zxm。在本例中，使用 edu.hebust.zxm.myfirstapplication 来作为包名。

Minimum Required SDK：应用程序所支持的最低 API 版本。为了支持尽可能多的设备，应把这个版本号设置成应用程序提供的核心功能所能使用的最低版本。如果应用程序有一些功能只在较新的 Android 版本才可用，并且它们不是应用程序的核心功能，那么可以在运行时，在支持这些功能的版本上启用这些功能。本例中选择 API 14，则这个应用只能运行在 Android SDK 4.0 以上版本的机器上。

Target SDK：应用程序针对的目标设备的平台版本，默认它被设置为 SDK 中最新可用的 Android 版本。

Compile With：编译应用程序所使用的平台版本，默认它被设置为 SDK 中最新可用的 Android 版本。

步骤 4：单击 Next 按钮。在接下来的对话框中设置应用程序的启动图标和 Activity，如图 2-12 所示。

Create Activity 选项用于设置是否让 ADT 自动创建一个默认的继承自 Activity 的类。该类是一个启动和控制程序的类，主要用来创建窗口 Activity。

如果勾选了 Create custom launcher icon 复选框，则会弹出如图 2-13 所示的对话框，可在其中设置应用程序的图标，否则会跳过这个对话框，采用默认的图标。

步骤 5：单击 Next 按钮。在弹出的对话框中选择一个 Activity 模板，如图 2-14 所示，开始构建应用程序。本例中，选择 Blank Activity。

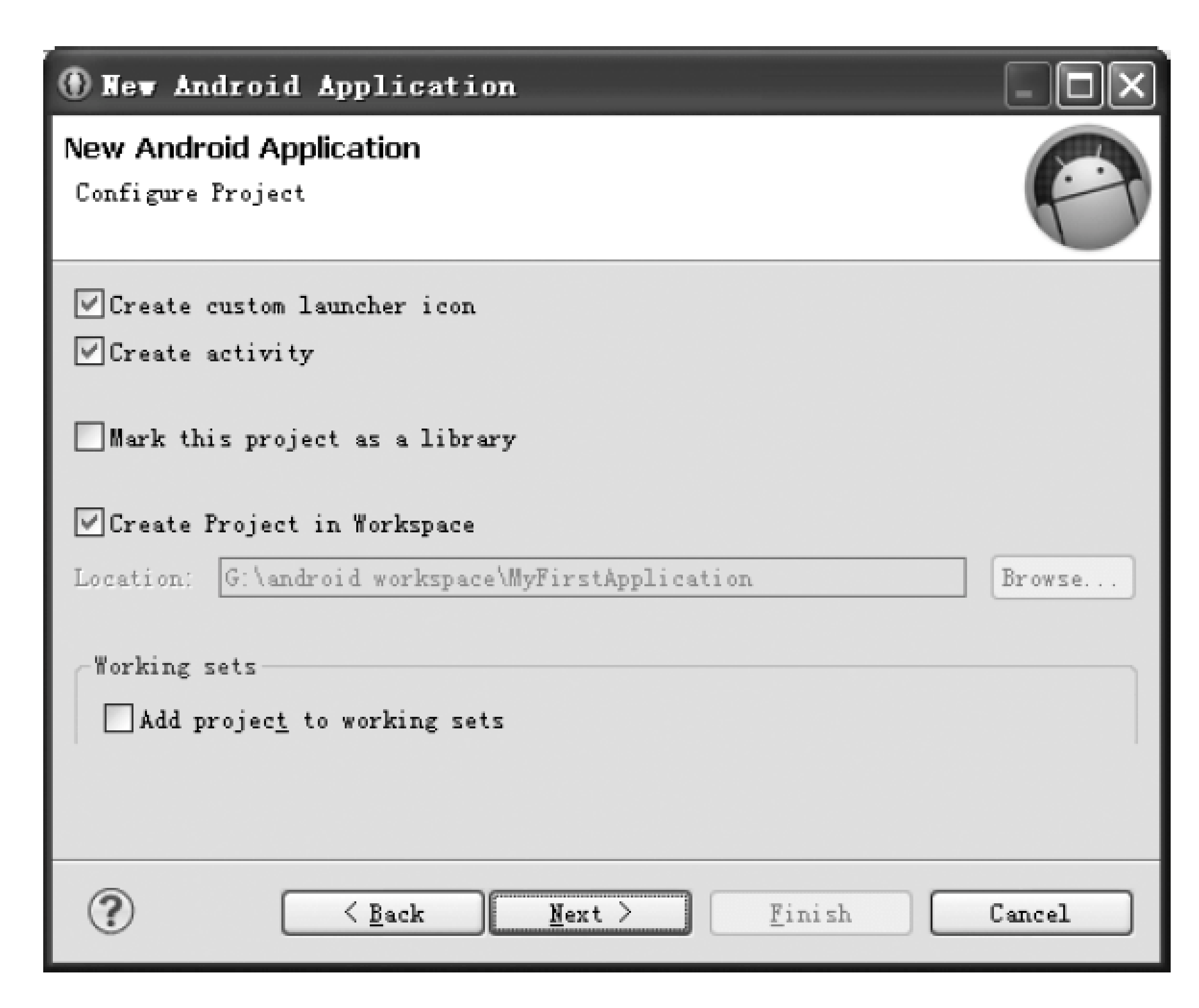


图 2-12 新建工程对话框(二)

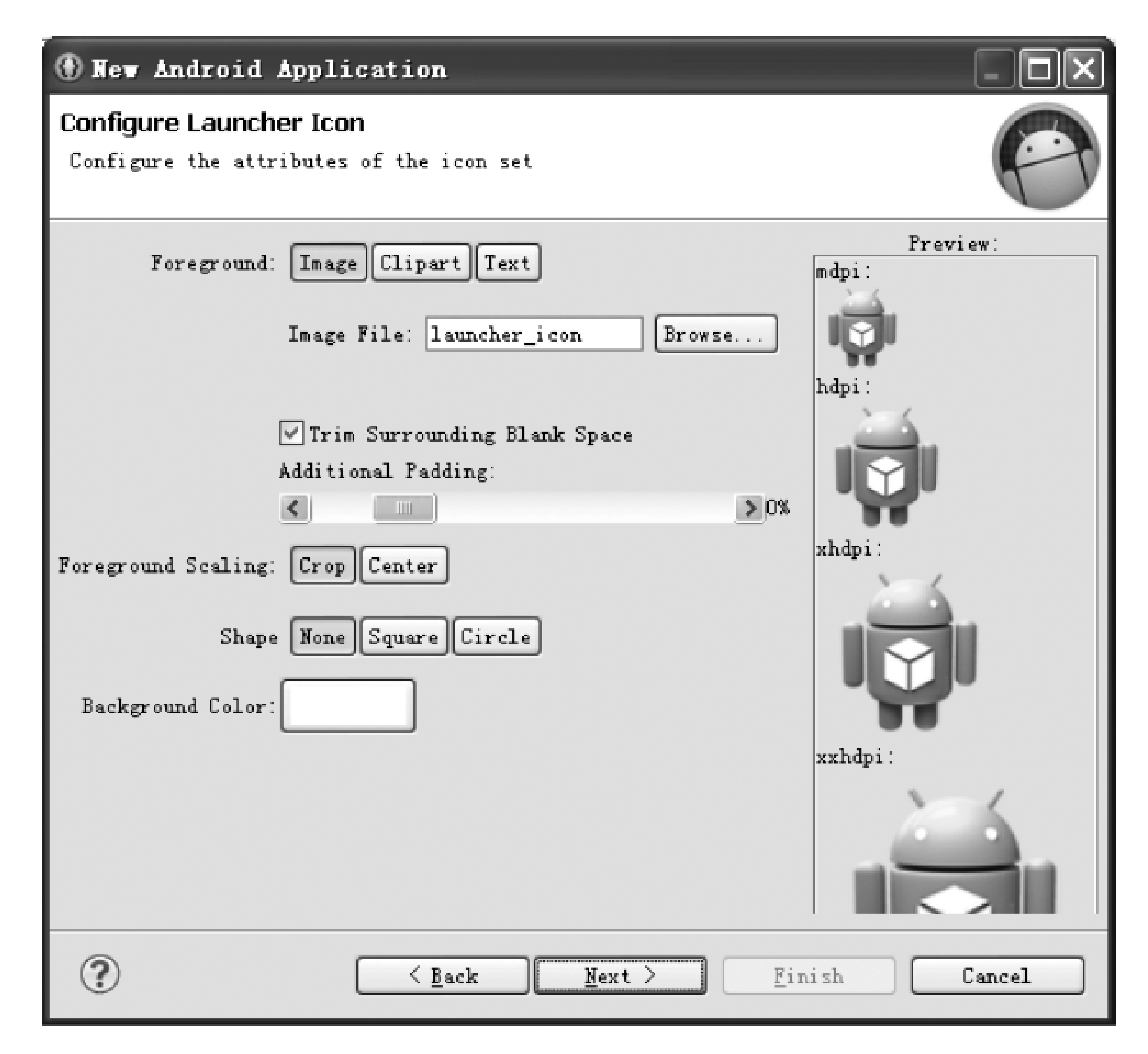


图 2-13 新建工程对话框(三)



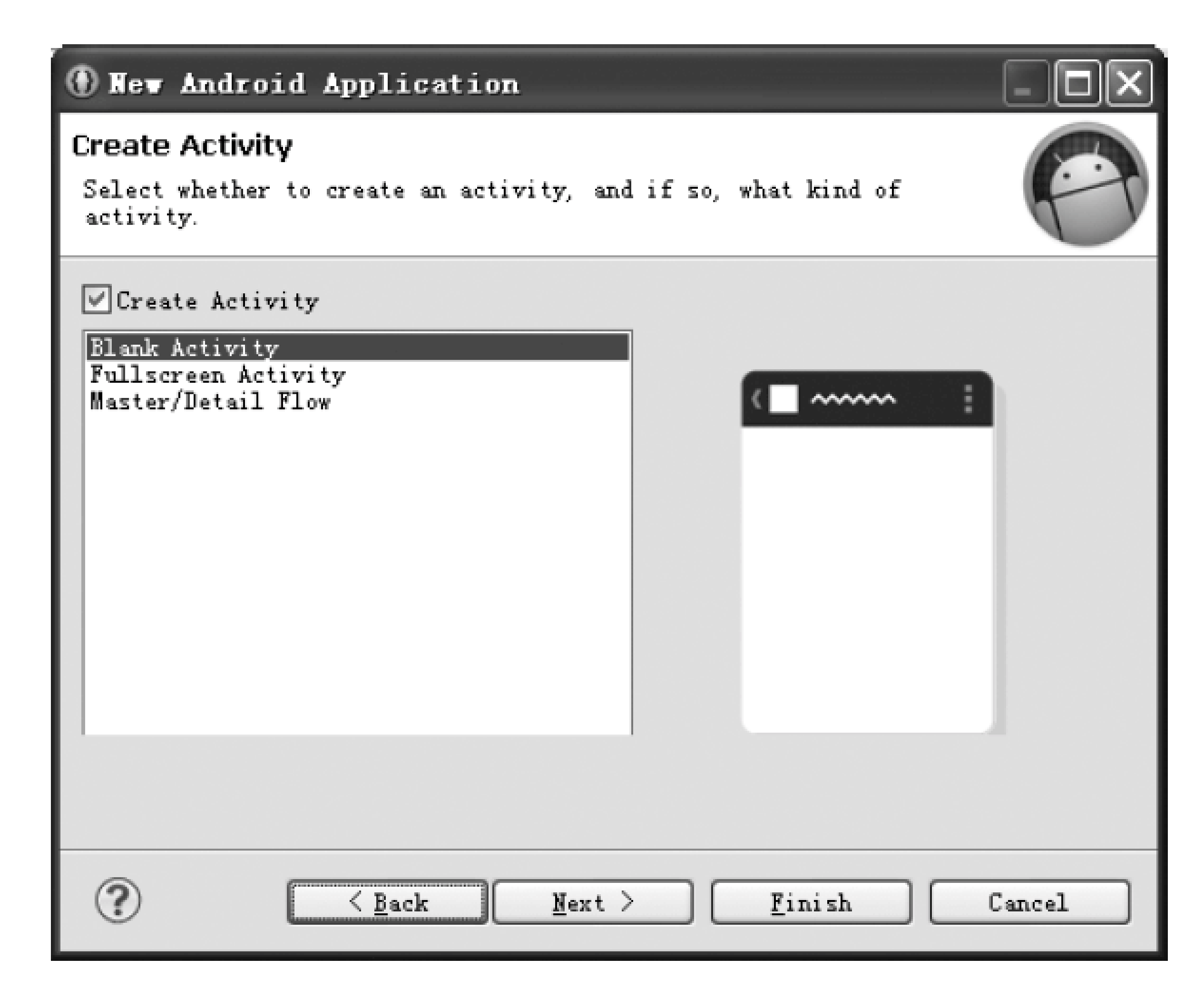


图 2-14　新建工程对话框(四)

步骤 6：单击 Next 按钮，弹出如图 2-15 所示的对话框。在该对话框中可以设置 Activity 的名称。将来 Android 系统运行该程序时，就是以这个 Activity 名称来辨别程序是否处于启动、暂停、继续还是关闭状态，以此来完成一个程序的活动周期。

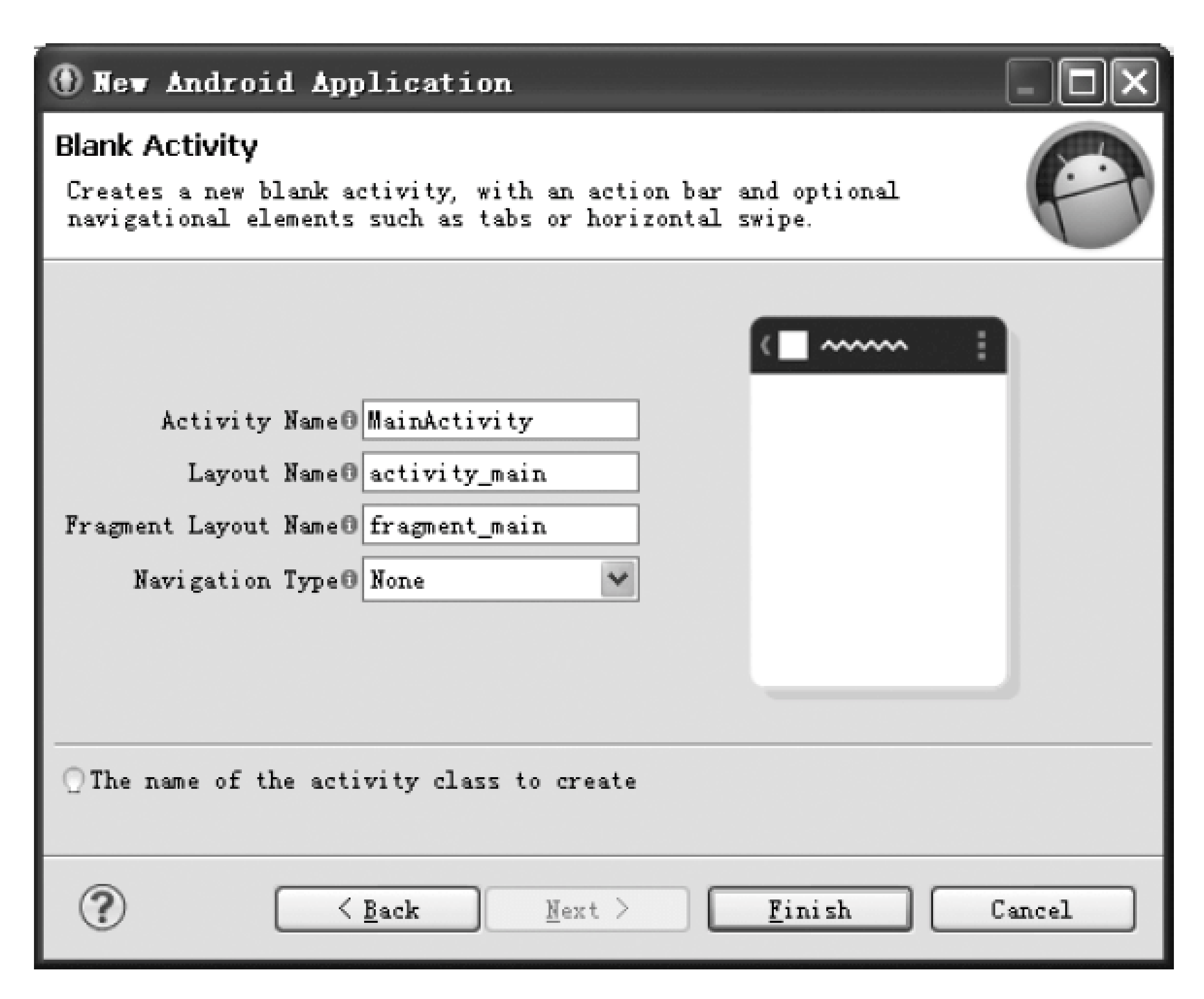


图 2-15　新建工程对话框(五)

本例中保留所有的默认设置，并单击 Finish 按钮，完成工程项目的创建。

这样，就成功建立了一个带有一些默认文件的 Android 项目，并且已经可以编译和运行该应用程序了。在 Eclipse 窗口左侧的 Package Explorer 面板中可以看到这个工程项目及其目录结构。

222 编译和运行 Android 应用程序

虽然在 2.2.1 节创建应用程序 MyFirstApplication 的步骤中没有编写任何程序代码，但这个 Android 项目中已经包含了一些默认文件，可以对其编译和运行。

1. 在模拟器上运行 Android 应用程序

运行 Android 程序时，可以在 Package Explorer 面板中右击工程名称，在弹出的快捷菜单中执行 Run As→Android Application 命令，Java 源码就会被编译，并被安装到虚拟设备上运行。

在模拟器上可以打开已安装到设备上的应用程序列表，如图 2-16 所示。在其中可以看到示例程序 MyFirstApplication。程序的运行结果如图 2-17 所示，单击模拟器右侧的 Home 按钮或返回按钮，可以终止应用程序的运行。



图 2-16 安装到应用程序列表中的示例程序

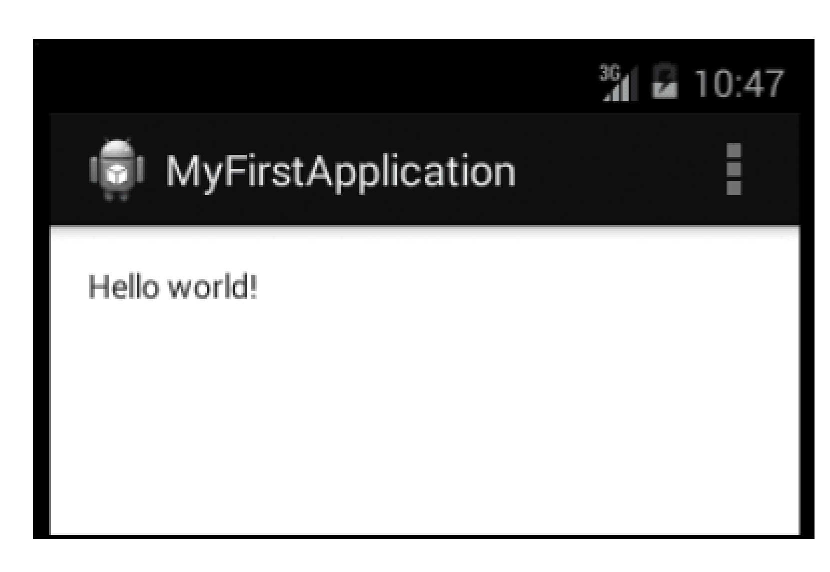


图 2-17 示例程序的运行结果

2. 在真实设备上运行 Android 应用程序

以手机为例，在真实设备上运行 Android 应用程序要首先安装设备的 USB 驱动程序。驱动程序安装好后，在计算机上插入手机，计算机就会显示设备已识别。不同的 Android 手机有不同的驱动和安装方式，有些直接用 Android SDK and AVD Manager 安装，有些需要去手机公司的网站上下载驱动程序，具体操作方法可参阅手机附带的手册，在此不再赘述。

设备和计算机正确连接后，设置 Android 手机为 USB 调试模式。具体步骤：打开菜单，选择“系统设置”→“开发者选项”→“USB 调试”选项。

在没有连接真实设备时，在 Eclipse 中右击工程，在弹出的快捷菜单中可以执行 Run As→Android Application 命令，程序会自动发送到模拟器运行。正确安装好手机的 USB 驱动程序，并连接真实设备之后，程序就会发送到真实设备上运行，效果与在模拟器中的一样。在真机上运行的速度一般比用模拟器要快很多。

如果程序仍然在模拟器而不是真实设备上运行，可以做如下处理：在 Eclipse 中右击工程，在弹出的快捷菜单中执行 Run As→>run configurations→target 命令，去掉模拟器选项的勾选，再重新运行程序。

无论采用哪种方式运行程序，编译器都会将所有编译生成完成的资源文件打包到

APK 文件，包括 assets 目录、res 目录、资源项索引文件 resources. arsc、应用程序的配置文件 AndroidManifest. xml，以及应用程序代码文件 classes. dex、用来描述应用程序的签名信息的文件等。这个 APK 文件可以直接拿到模拟器或者设备上去安装运行。

Android 工程文件夹下的 bin 文件夹在编译成功后会生成. apk 文件，本例中生成的是 MyFirstApplication. apk，如图 2-18 所示。这就是可安装的 Android 程序，可以用任何移动设备的同步工具像安装其他 Android 程序一样安装自己的项目。之后在 Android 系统的应用程序目录下找到该项目图标，就可以在设备上运行了。

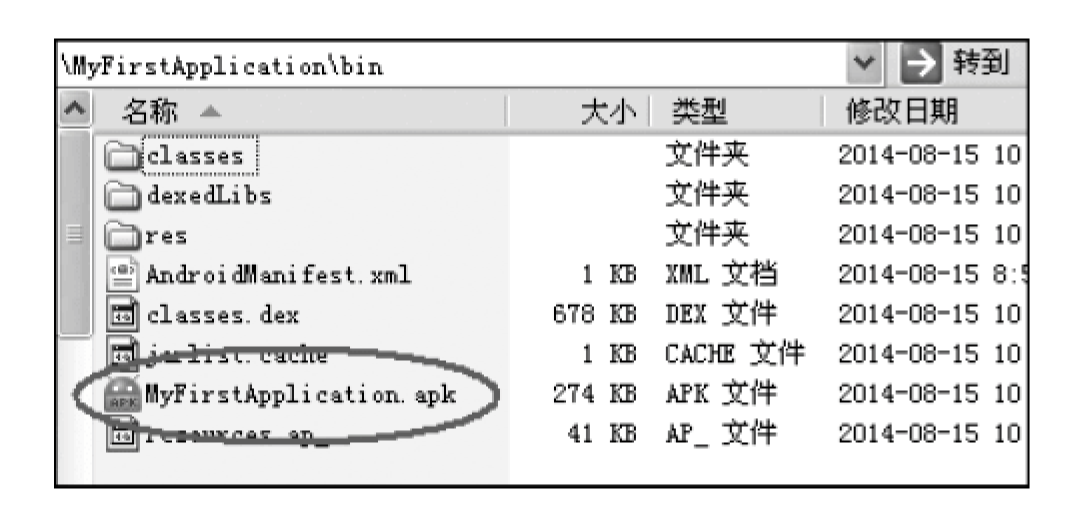


图 2-18　可安装的 Android 程序

223　移动设备上应用程序的卸载

卸载真实设备上应用程序的方法与模拟器类似，在此以模拟器上操作为例。

单击模拟器右侧的 MENU 按钮，单击弹出的“管理应用(Manage apps)”菜单项，打开应用程序列表，在所列的所用应用程序中选择自己要删除的程序，如图 2-19(a)所示。打开该应用程序的“应用信息”界面，如图 2-19(b)所示。如果程序正在运行，可先单击“强行停止”按钮，停止程序，然后单击“卸载”按钮，弹出“确认”对话框，单击“确定”按钮，就把该程序从模拟器上删除了。



(a)



(b)

图 2-19　卸载应用程序

2.3 Android 工程项目的文件构成

2.3.1 工程项目的目录结构

Android 工程项目文件夹包含了组成 Android 应用程序源代码的所有文件。在 Eclipse 窗口左侧的 Package Explorer 面板中可以展开工程项目文件夹，如图 2-20 所示，可以看到 MyFirstApplication 项目包含 src、gen、assets、res 等目录。通常，一个标准的 Android 应用程序的工程项目包含 src 文件夹、gen 文件夹、libs 文件夹、assets 文件夹、res 文件夹、应用配置文件 AndroidManifest.xml、default.properties 等。

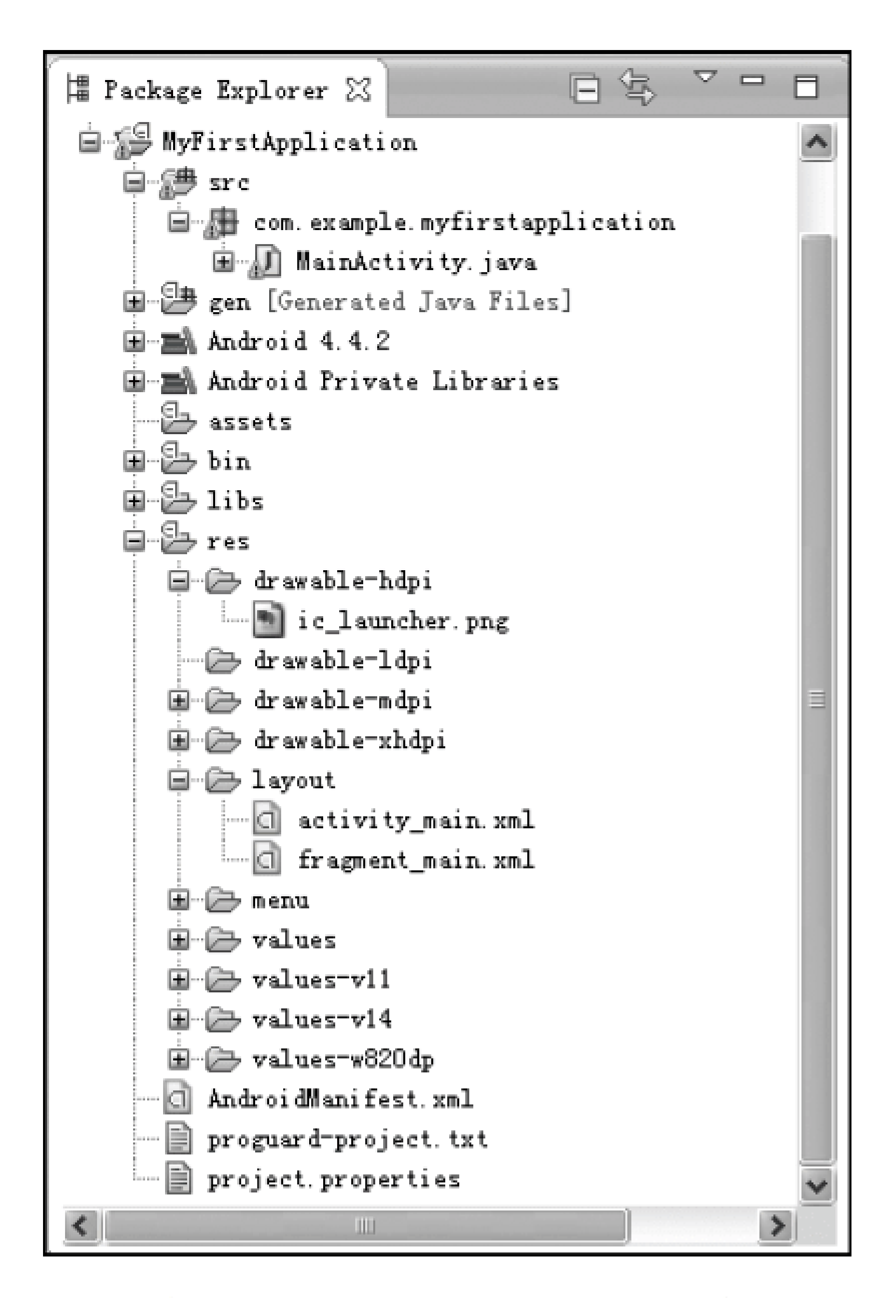


图 2-20 Package Explorer 面板

2.3.2 源码文件夹 src 和 gen\R.java

src 文件夹是应用程序源代码所在的文件夹，该文件夹为必需的。本章示例中，该文件夹中有文件 MainActivity.java，这是一个定义了 Android 应用程序入口的源文件。通常一个 Android 应用程序的程序逻辑以及功能代码都是写在该目录下的，不同功能的类可以通过 Java 包的机制来进行区分。文件夹的内部结构根据用户所声明的包自动组织，包名就是在新建工程时指定的 Package Name，包的作用就像文件夹一样，便于分门别类地管理程序。

本章的示例工程由于在创建时勾选了 Create Activity 选项，所以在该目录下生成了继承自 Activity 的启动与控制程序的类 MainActivity，打开自动生成的 MainActivity.

java 源文件，其主要代码如下：

```
package edu.hebust.zxm.myfirstapplication;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

gen 文件夹是应用程序自动生成的资源文件 R.java 所在的文件夹，该文件夹也是必需的。其中的 R.java 文件由 Eclipse 自动生成与维护，提供了对 Android 资源的全局索引。

Android 应用程序中，XML 布局和资源文件并不包含在 Activity 的 Java 源码中，各种资源文件由系统自动生成的 R.java 文件来管理。每一个资源类型在 R.java 文件中都有一个对应的内部类，例如，类型为 layout 的资源项在 R.java 文件中对应的内部类为 layout，而类型为 string 的资源项在 R.java 文件中对应的内部类就为 string。R.java 文件的作用相当于一个项目字典，项目中的用户界面、字符串、图片、声音等资源都会在该类中创建其唯一的 ID，当项目中使用这些资源时，会通过该 ID 得到资源的引用。如果程序开发人员变更了任何资源文件的内容或属性，R.java 文件会随之变动并自动更新 R.java 类。

可以打开 R.Java 文件查看其内容，但开发者不需要也不能修改此文件，否则资源的内存地址会发生错误，程序就无法运行了。如果在资源目录中增加或删除了资源文件，可以在工程名称上右击，选择 Refresh 命令来更新 R.java 文件中的代码。

在 Java 程序中通过 R.java 类引用资源的方法是“R.资源类型.资源名称”，其中的“资源类型”可以是放置图像的文件夹、XML 文件或布局文件，而“资源名称”是资源名或 XML 文件中的文件或变量名。例如，R.drawable.background 表示使用资源目录中的 res\drawable\background.png 图片文件；R.string.title 表示使用资源文件 res\values\string.xml 中定义的 title 字符串变量；R.layout.activity_main 表示使用资源目录中的 res\layout\activity_main.xml 布局文件；R.anim.anim 表示使用 res\anim\anim.xml 动画定义文件。

2.3.3　Android.jar 文件夹

在应用程序中，一般有诸如 Android2.2、Android4.4.2 等文件夹，在其下面的 Android.jar 文件是支持该应用程序运行的 JAR 包，它同时还包含该应用程序打包 APK

文件时需要的 META-INF 目录。同时它包含 Android SDK 架构中 Android RunTime 层中与 Dalvik 虚拟机同处一层的 Android Core Library。这个 JAR 文件是应用程序的基础。

2.3.4 资源文件夹 res 和布局文件

res 文件夹是所有应用程序资源所在的文件夹,该文件夹为必需的。应用程序资源包括动画、图像、布局文件、XML 文件、数据资源(如字符串)和原始(raw)文件。该文件夹按照资源的种类默认分为 3 个子文件夹,分别为 drawable、layout 和 values。通常,drawable 文件夹中主要存放的是一些图片格式文件,支持的格式有.png、.jpg 等位图文件;layout 文件夹中主要存放的是界面布局的 XML 文件;values 文件夹中包含了所有的 XML 格式的参数描述文件,如 string.xml(字符串描述文件)、color.xml(颜色描述文件)、style.xml(样式描述文件)和 array.xml(数组描述文件)等。

在 Android 的应用程序中的用户界面有两种生成方式。可以采用 XML 布局文件来指定用户界面,也可以采用在 Activity 中书写 Java 代码的方式来设定用户界面。通常使用前一种方法,以下是一个 XML 布局文件的示例,文件名为 main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv_message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="布局文件示例\n"    />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"    >
        <Button android:id="@+id/btn_1"
            android:text="开始"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <Button android:id="@+id/btn_2"
            android:text="结束"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
</LinearLayout>
```



上述 XML 布局文件中第 1 行声明了文档的版本以及编码方式，第 2 行中的 LinearLayout 标签声明了界面的布局方式为线性布局，xmlns:android 属性声明了使用的 Android 命名空间，这个属性是必需的。第 3 行则声明了界面的布局是垂直布局，第 4 行与第 5 行描述布局的宽和高将充满整个父容器。在该界面中只定义了一个 TextView 控件和两个 Button 控件。两个 Button 控件使用水平线性布局。

在 Activity 代码中调用 setContentView(R.layout.main)方法，就可以在界面中显示出由 res/layout/main.xml 这个文件设置的 Activity 外观，运行结果如图 2-21 所示。



图 2-21　示例布局文件的显示结果

在 XML 文件中可以修改或添加欲显示的 Widget 控件，如在本例添加了显示文字的 TextView 控件，并指定了其文字宽、高，以及显示的文字字符串内容。在添加 Button 控件时，为了方便日后在 Activity 中能捕获到针对这个按钮的动作，需要在“<Button”标签后使用 ID 资源属性来声明相应的 Button 的 ID 号，格式为：android:id="@+id/资源名称"，其中“@+id/”表明为资源添加一个 ID 号，这样才可以让应用程序中的 R.java 类资源文件以及 Activity 文件能正确地记录和引用该资源。

在 XML 布局文件中并没有具体的处理逻辑，如按下按钮后的动作，也没有需要生成的事件或动作。它的作用仅仅是将控件显示到窗口中。

在图 2-20 中，layout 文件夹中有 activity_main.xml 文件，这是 Eclipse 为 Android 应用程序设置的默认的主窗口界面布局文件，在创建项目时可以设置这个文件的文件名。用户可根据需要新建多个 XML 布局文件。在一个工程中，可以为不同的 Activity 指定不同的 XML 布局文件，它们就会有不同的用户界面。

另外一个常用的资源文件是字符串资源描述文件 string.xml，一般位于工程 res 文件夹的 values 子文件夹下。下面的代码是一个典型的 string.xml 代码段。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MyFirstApplication</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

上述代码段的第 1 行定义了 XML 版本与编码方式；第 2 行以后使用<resources>标签定义了两个字符串，字符串的名称分别为 app_name 和 hello_world。如果需要在程序代码中使用这些字符串，可以用“R.string.字符串名称”的方式引用。

res 目录下的 anim、xml、raw 文件夹并不是 Android 程序默认生成的，这 3 个目录一般放置动画文件 anim.xml、其他用途的 XML 文件、音效文件等。

2.3.5　assets 文件夹

同 res 文件夹相似，assets 也是存放资源文件的文件夹，但与 res 文件夹的不同之处

在于 res 文件夹中的内容会被编译器编译，而 assets 文件夹中的内容则不会被编译器编译。也就是说应用程序运行时，res 文件夹中的内容会在启动时载入内存，assets 文件夹中的内容只有在被用到时才会载入内存，所以一般将一些不经常使用的大资源文件存放在该目录下。例如，应用程序中使用到的音频或者视频文件、图片和文本文件等。

在程序中可以使用 getResources. getAssets(). open("文件名")的方式得到资源文件的输入流 InputStream 对象。

对于本例，由于是刚刚创建的工程，还没有放置任何资源文件进去，所以 assets 文件夹是空的。

2.3.6 应用配置文件 AndroidManifest.xml

AndroidManifest. xml 是全局应用程序描述文件，它定义了应用程序的能力(capability)和权限，以及运行方式。

Android 程序必须在根目录下包含一个 AndroidManifest. xml 文件。AndroidManifest. xml 定义了应用程序的整体布局、提供的内容与动作，还描述了程序的信息，包括应用程序的包名、所包含的组件、服务(Service)、接收器(Receiver)、应用程序兼容的最低版本、图标、应用程序自身应该具有的权限的声明以及其他应用程序访问该应用程序时应该具有的权限等。它是应用程序的重要组成文件。

以下是一个 AndroidManifest. xml 文件的示例：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hebust.zxm.layoutexample"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="19" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="edu.hebust.zxm.layoutexample.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```




```
</manifest>
```

AndroidManifest.xml 文件的根元素是<manifest>，它包含 xmlns:android、package、android:versionCode 和 android:versionName 共 4 个属性。本例中的第 1 行声明了 XML 的版本以及编码方式，第 2 行声明了命名空间 android，自此以后所有的 android 变量都将代表 http://schemas.android.com/apk/res/android，引用 Android 系统提供的 UI 资源。第 3 行声明了主程序所在的包名，android:versionCode 定义了应用程序的版本号，数值越大说明版本越新，但仅在程序内部使用，并不提供给应用程序的使用者；android:versionName 定义了应用程序的版本名称，是一个字符串，仅限于为用户提供一个版本标识。第 7 行和第 8 行则声明了 Android SDK 的版本信息。

第 9 行开始定义<application>元素。<manifest>根元素仅能包含一个<application>元素，<application>元素中声明 Android 程序中的组成部分，包括 Activity、Service、Broadcast Receiver 和 Content Provider，所定义的属性将影响所有组成部分。

其中 icon 属性指出了应用程序安装完后的桌面图标；label 属性指出了应用程序的标签文字，本例中分别通过@符号引用了 res/drawable 目录下的 ic_launcher.png 图片和 res/values 目录下名称为 app_name 的字符串，这种引用方式也是实际编程中常用的一种方法。

在<application>元素中要声明程序运行过程中用到的 Activity 类，本例中声明了一个 Activity 类，即 MainActivity，其<intent-filter>子元素的属性指出该 Activity 是程序启动时第一个启动的窗口。当 Activity 动作发生时它会创造一个 Intent 对象，这个 Intent 对象抽象地描述了 Activity 想要进行的动作，Intent 对象可以包含操作 Activity 时所要提供的数据或消息，共有如下几种形式：action（动作）、data（信息）、category（种类）。不同的应用程序有不同的 Intent 对象，所以要通过一个 intent-filter 来筛选最适当的数据或消息。本例使用<action>名称定义了由 android.intent.action 类进行 MAIN 动作，表示 Activity 的启动，无任何信息输出；使用类 android.intent.category.LAUNCHER 启动这个程序，这也是 Intent 的另一种形式。

如果工程中有多个 Activity，则要手动到<application>元素中添加声明，格式如下：

```
<activity android:name="包名.Activity名称" />
```

在应用程序的 AndroidManifest.xml 文件中还可以为应用程序指定相应的权限，如网络权限、短信权限、电话权限等。应用程序的所有权限全部封装在 android.Manifest 这个类中。

权限的使用方法是将权限声明语句添加在 AndroidManifest.xml 文件中。应用程序的权限标签是<application></application>的兄弟标签，通常写在</application>后面、</manifest>标签之前，例如某个应用程序需要添加发短信的权限时的声明：

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

应用程序除了声明自身应该具有的权限外，还可以声明访问本程序的应用所应当具

有的权限，例如要求其他应用程序访问本应用程序时应该具有 SEND_SMS 权限时，在<activity>标签中添加权限：

```
<activity
    android:name=".Activity"
    android:label="@string/app_name">
    <uses-permission android:name="android.permission.SEND_SMS" />
</activity>
```

2.3.7 default.properties 文件

另一个和 AndroidManifest.xml 文件有关的是 default.properties 文件，它是自动创建的工程文件，定义了程序所使用的 Android SDK 版本，即应用程序针对的目标设备以及相应的 API Level。该文件为项目的配置文件，不需要人为改动，系统会自动对其进行管理。

2.4 Android 应用的组成

2.4.1 Android 应用的基本组件

一般来说，Android 应用程序由 Activity 及 Activity Manager、ContentProvider、Service、BroadcastReceiver 等部分组成。当然，有些应用程序可能只包含其中部分而非全部。它们在 AndroidManifest.xml 配置文件中以不同的 XML 标签声明后，才可以在应用程序中使用。

Activity 一般含有一组用于构建用户界面(UI)的 Widget 控件，如按钮 Button、文本框 TextBox、列表 List 等，实现与用户的交互，相当于 Windows 应用程序的对话框窗口或网络应用程序的 Web 页面窗口；Activity Manager 用于管理应用程序的生命周期。一个功能完善的 Android 应用程序一般由多个 Activity 构成，这些 Activity 之间可互相跳转，可进行页面间的数据传递。例如，显示一个 E-mail 通讯簿列表的界面就是一个 Activity，而编辑通讯簿界面则是另一个 Activity。

ContentProvider 是 Android 系统提供的一种标准的共享数据的机制。在 Android 平台下，一个应用程序使用的数据存储都是私有的，其他应用程序是不能访问和使用的。私有数据可以是存储在文件系统中的文件，也可以是 SQLite 中的数据库。当需要共享数据时，ContentProvider 提供了应用程序之间数据交换的机制。一个应用程序通过实现一个 ContentProvider 的抽象接口将自己的数据暴露出去，并且隐蔽了具体的数据存储实现，这样既实现了应用程序内部数据的保密性，又能够让其他应用程序使用这些私有数据。一个 ContentProvider 提供了一组标准的接口，能够让应用程序保存或读取各种数据，同时实现了权限机制，保护了数据交互的安全性。

Service 是与 Activity 独立且可以保持后台运行的服务，相当于一个在后台运行的没有界面的 Activity。如果应用程序并不需要显示交互界面但却需要长时间运行，就需要

使用 Service。例如，在后台运行的音乐播放器，为了避免音乐播放器在后台运行时该程序被终止而停播，需要为其添加 Service，通过调用 Context.startService()方法，让音乐播放器一直在后台运行，直到使用者再调出音乐播放器界面并关掉它为止。用户可以通过 StartService()方法启动一个 Service，也可通过 Context.bindService()方法来绑定一个 Service 并启动它。

在 Android 中，广播是一种广泛运用的在应用程序之间传输信息的机制。而 BroadcastReceiver 是用来接收并响应广播消息的组件，不包含任何用户界面。可以通过启动 Activity 或者 Notification 通知用户接收到重要信息。Notification 能够通过多种方法提示用户，包括闪动背景灯、震动设备、发出声音或在状态栏上放置一个持久的图标。

Activity、Service 和 BroadcastReceiver 都是由 Intent 异步消息激活的。Intent 用于连接以上各个组件，并在其间传递消息。例如，广播机制一般通过下述过程实现：首先在需要发送信息的地方，把要发送的信息和用于过滤的信息(如 Action、Category)装入一个 Intent 对象，然后通过调用 Context.sendBroadcast()、sendOrderBroadcast()或 sendStickyBroadcast()方法，把 Intent 对象以广播方式发送出去。Android 使用 intent-filter 来处理对这种广播信息的接收。当 Intent 发送以后，所有已经注册的 BroadcastReceiver 会检查注册时的 IntentFilter 是否与发送的 Intent 相匹配，若匹配则就会调用 BroadcastReceiver 的 onReceive()方法，对其接收并响应。例如，对于一个电话程序，当有来电时，电话程序就自动使用 BroadcastReceiver 取得对方的来电消息并显示。使用 Intent 还可以方便地实现各个 Activity 间的跳转和参数传递。

2.4.2　什么是 Activity

Activity 是 Android 程序的呈现层，显示可视化的用户界面，并接收与用户交互所产生的界面事件。Android 应用程序可以包含一个或多个 Activity，一般在程序启动后会首先呈现一个主 Activity，用于提示用户程序已经正常启动并显示一个初始的用户界面。Activity 在界面上的表现形式有全屏窗体、非全屏悬浮窗体、对话框等。

对于大多数与用户交互的程序来说，Activity 是必不可少，也是非常重要的。刚开始接触 Android 应用程序时，可以暂且将 Activity 简单地理解为用户界面。新建一个 Android 项目时，系统默认生成一个启动的 Activity，其默认的类名为 MainActivity，源码文件中的主要内容如下：

```
import android.app.Activity;                //每一个 Android的 Activity都需要继承自 Activity类
import android.os.Bundle;                       //用于映射字符串值

public class MainActivity extends Activity {
                                            //MainActivity是名称,其父类是 Activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
            //设置布局,它调用了 res/layout/ activity_main.xml 中定义的界面元素
}
```

应用程序中的每个 Activity 都继承于 android.app.Activity 类并重写其 OnCreate()方法,该类是 Android 提供的基类,继承该父类后,通过重写(Override)父类的方法来实现各种需要的功能。

程序启动后显示的第一个界面是应用程序的第一个 Activity,而后可以根据需要从这个 Activity 启动另一个新的 Activity。上例中,程序启动后显示的第一个界面是 MainActivity。@Override 表示重写父类的 onCreate()方法,Bundle 类型的参数保存了应用程序上次关闭时的状态,并且可以通过一个 Activity 传给下一个 Activity;在 Activity 的生命周期中,只要离开了可见阶段(即失去了焦点),它就很可能被进程终止,这时就需要有种机制,能保存当时的状态,这就是其参数 savedInstanceState 的作用。有关 Bundle 的细节详见后续章节。

Activity 类通常要与布局资源文件(res/layout 目录下的 XML 文件)相关联,并通过 setContentView()方法将布局呈现出来。在 Activity 类中包含控件的显示、界面交互设计、事件的响应设计以及数据处理设计、导航设计等内容。

一个功能完善的 Android 应用程序一般由多个 Activity 构成,这些 Activity 之间可互相跳转,可进行页面间的数据传递。例如,一个 E-mail 程序可能包含 3 个 Activity:显示邮件列表的 Activity、显示邮件内容的 Activity、写新邮件或回复邮件的 Activity。

需要注意的是,应用程序中的所有 Activity 都必须在 AndroidManifest.xml 文件中添加相应的声明,并设置其属性和<intent-filter>。如下的 AndroidManifest.xml 代码片段中就含有对两个 Activity(MainActivity 和 SecondActivity)的声明。

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity
        android:name="com.example.myfirstapplication.MainActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="com.example.myfirstapplication.SecondActivity"
        android:label="SecondActivity">
    </activity>
</application>
```

上述代码中有两个＜activity＞＜/activity＞元素：第一个为系统默认生成的 MainActivity；第二个为用户新建的 SecondActivity，其中 MainActivity 是程序入口。

当一个应用程序包含多个 Activity 时，从一个 Activity 可以切换到另一个 Activity，Activity 之间切换时，需要使用 Intent 对象。关于 Intent 的相关知识将在第 6 章介绍。

2.4.3 Activity 的生命周期

所有 Android 组件都具有自己的生命周期，生命周期是指从组件建立到组件销毁的整个过程。在生命周期中，组件会在可见、不可见、活动、非活动等状态中不断变化。

Activity 具有自己的生命周期，其生命周期指 Activity 从启动到销毁的过程。生命周期由系统控制，程序无法改变，但可以用 onSaveInstanceState()方法保存其状态。了解 Activity 的生命周期有助于理解 Activity 的运行方式和编写正确的 Activity 代码。

Activity 在生命周期中表现为 4 种状态，分别是活动状态、暂停状态、停止状态和非活动状态。活动状态时，Activity 在用户界面中处于最上层，完全能被用户看到，能够与用户进行交互。暂停状态时，Activity 在界面上被部分遮挡，该 Activity 不再处于用户界面的最上层，且不能够与用户进行交互。停止状态时，Activity 在界面上完全不能被用户看到，也就是说这个 Activity 被其他 Activity 全部遮挡。非活动状态指不在以上 3 种状态中的 Activity。

Activity 的生命周期如图 2-22 所示。

(1) 启动 Activity。系统会先调用 onCreate()方法，然后调用 onStart()方法，最后调用 onResume()方法，Activity 进入活动状态。

(2) 当前 Activity 被其他 Activity 部分覆盖或被锁屏，Activity 不能与用户交互。系统会调用 onPause()方法，暂停当前 Activity 的执行，Activity 进入暂停状态。

(3) 当前 Activity 由被覆盖状态回到前台或解锁屏。系统会调用 onResume()方法，再次进入活动状态。

(4) 当前 Activity 转到新的 Activity 界面或按 Home 键回到主屏幕，当前 Activity 完全不可见，转到后台。系统会先调用 onPause()方法，然后调用 onStop()方法，进入停止状态。

(5) 当前 Activity 处于停止状态时，用户后退回到此 Activity。系统会先调用 onRestart()方法，然后调用 onStart()方法，最后调用 onResume()方法，Activity 再次进入活动状态。

(6) 当前 Activity 处于被覆盖状态或者后台不可见，即处于暂停状态或停止状态时，如果系统内存不足，就有可能杀死这个 Activity。然后用户如果退回到这个 Activity，则会再次调用 onCreate()方法和 onStart()方法、onResume()方法，使其进入活动状态。

(7) 用户退出当前 Activity。系统先调用 onPause()方法，然后调用 onStop()方法，最后调用 onDestory()方法，结束当前 Activity。

Activity 的生命周期可分为可视生命周期和活动生命周期，如图 2-22 所示。可视生命周期是 Activity 在界面上从可见到不可见的过程，开始于 onStart()，结束于 onStop()。活动生命周期是 Activity 在屏幕的最上层，并能够与用户交互的阶段，开始于 onResume()，结束

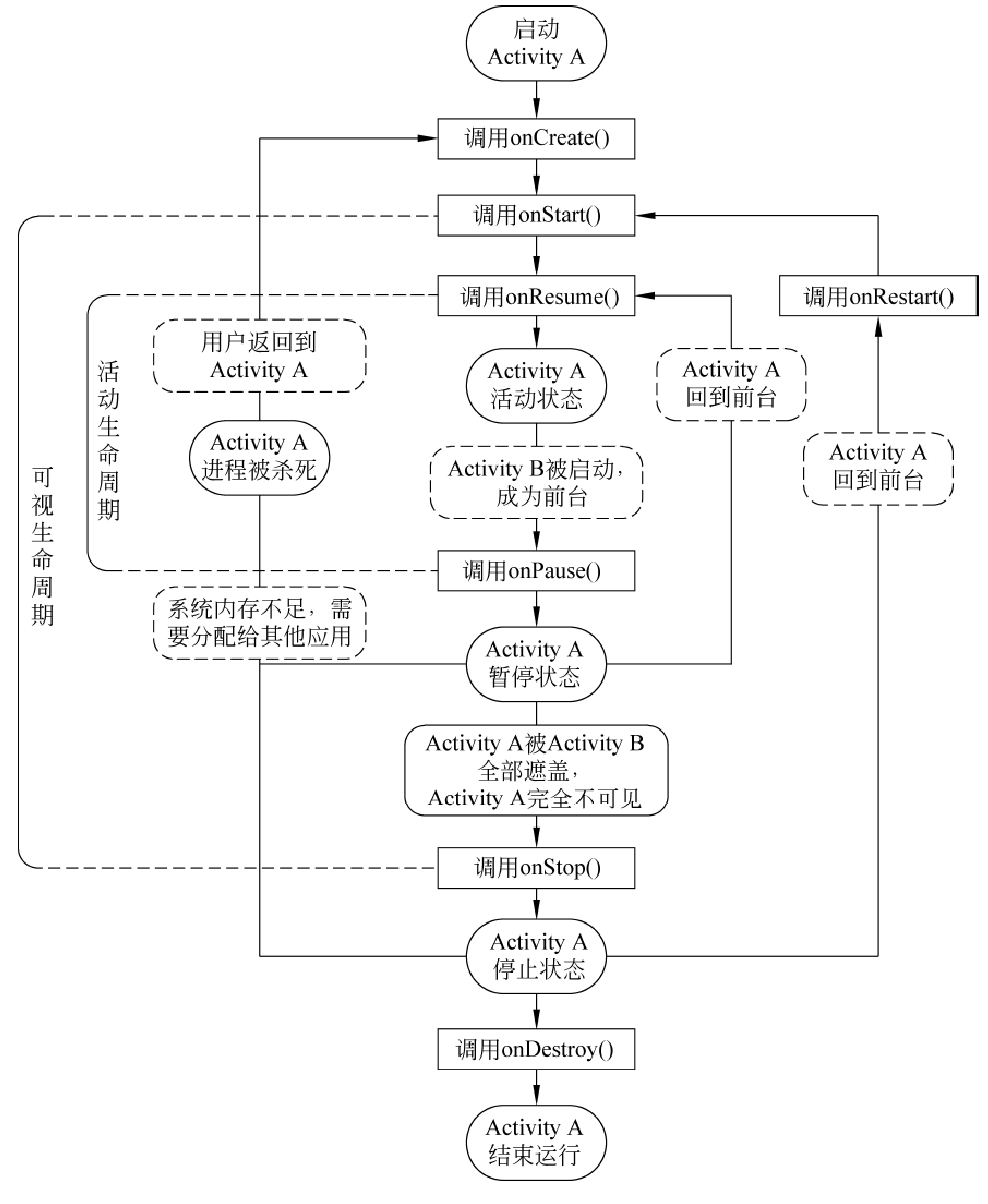


图 2-22　Activity 的生命周期示意图

于onPause()。在 Activity 的状态变换过程中 onResume()和 onPause()经常被调用，因此这两个方法中应使用简单、高效的代码。

在 Android 系统中，所有的 Activity 由堆栈进行管理，Activity 栈遵循“后进先出”的规则。如图 2-23 所示，当一个新的 Activity 被执行后，它将会被放置到堆栈的最顶端，并且变成当前活动的 Activity，而先前的 Activity 原则上还是会存在于堆栈中，但它此时不会在前台。Android 系统会自动记录从首个 Activity 到其他 Activity 的所有跳转记录并且自动将以前的 Activity 压入系统堆栈，用户可以通过编程的方式删除历史堆栈中的 Activity 实例。

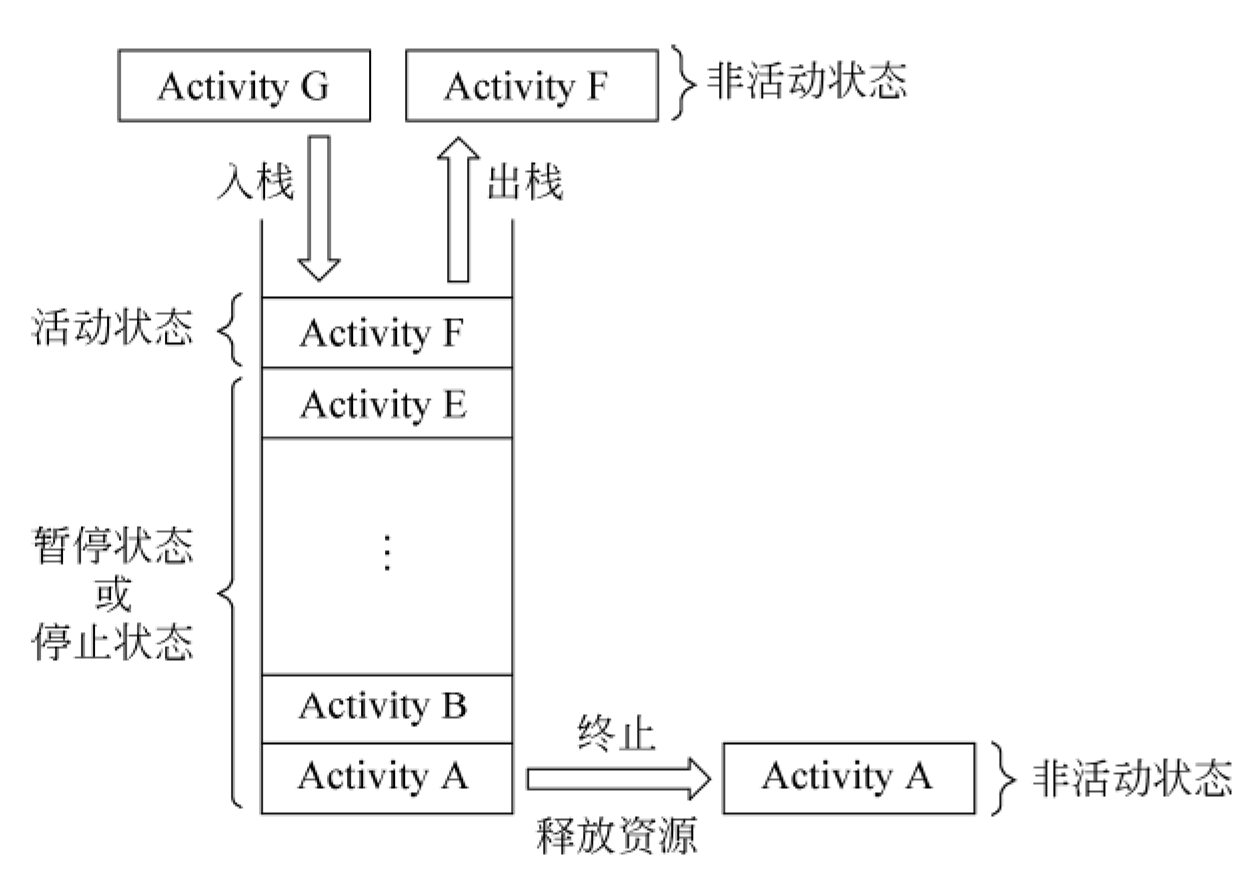


图 2-23　Activity 栈示意图

2.5　编写规范的 Android 代码

初学编程，一定要养成按照编码规范来进行编写程序的习惯。一个不按照编码规范编写的程序虽然能够正确运行，但它不易于阅读和维护，不是一个好程序。

编码规范包括很多内容，如文档的规范、代码的编写规则、命名规则和代码注释等。本节主要介绍常用的代码编写规则、命名规范和注释规范。

1. 代码编写规则

必须按照缩进的格式编写代码，缩进可以使用 Tab 键或者 4 个空格。在 Eclipse 中默认 4 个空格为一个 Tab 缩进单位。

要尽量避免一行的代码太长。当代码在一行中放不下时，应手动换行。要按照级别来进行换行，并且同级别对齐。

另外，段落之间可以使用空行间隔。

2. 命名规范

规范的命名使程序更易读，同时它们也可以提供一些有关标识符功能的信息，以助于理解代码。不论是一个常量、包，还是类，通常使用完整的英文描述来命名，同时避免超长的命名和相似的命名。例如，ActivityObject 和 ActivityObjects，最好不要一起使用。命名时要慎用缩写，如果要用到缩写，要按照通用缩写规则使用缩写，例如，No. 代表 number，ID. 代表 identification。

1）包(Package)的命名规则

在 Android 系统上安装的所有包中，自己的包名必须是唯一的。包名的前缀总是全部小写的 ASCII 字母。一般项目的包名以机构域名倒写开头，如 com. google。后面是程序所在项目的英文名称，通常不含版本号，除非有特别需要与以前版本区分，如两个版本

可能同时运行。再后面为子系统的名称，每个子系统内按照类别区分，如 com. google. widget. TimePicker。

2）类(Classe)和接口(Interface)的命名规则

对于所有的类来说，类名的首字母应该大写。通常类名是一个名词，如果类名由若干单词组成，那么每个单词的首字母应该大写，例如 MyFirstActivityClass。尽量使类名简洁而便于描述，使用完整单词，避免缩写。接口的大小写规则与类名相似，一般以 I(大写 i)开头，常以 able、ible 结尾。

3）方法(Method)的命名规则

通常方法名是一个动词，以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写，如 run()、runFast()、getBackground()。

4）变量(Variable)和参数(Parameter)

变量用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。尽管语法上允许，变量名通常不以下划线或美元符号开头。变量名应简短且便于描述。变量名的选用应该易于记忆，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。

5）集合(Collection)变量

集合变量，例如数组、向量等，在命名的时候应该从名字上面体现出该变量为复数，还可以使用 some 词头，如 customers、someMessages。

6）常量(Constant)

常量的声明，应该全部大写，单词间用下划线隔开，如 MIN_WIDTH。

3. 注释规范

注释就是在程序中给出一些解释，或提示某段代码的作用。注释是不被编译的，所以不用担心执行效率的问题。在 Java 中注释分为行注释、块注释和文档注释。

行注释就是一整行的注释信息，单行注释也是最常用的，行注释的符号是//，在注释符号后面一整行都被作为注释信息。

块注释以/ * 开始，以 * /结束，在这个区域内的文字都将作为注释信息。

文档注释通常是用来描述类/接口或方法的，一般写在类/接口定义或方法定义的前面。文档注释可以帮助程序员了解此类或方法具有哪些功能，需要什么样的参数等相关的信息。文档注释以/**开头，以 * /结尾。

类、接口的文档注释通常包含有关整个类或接口的信息，包括用途、如何使用、开发维护的日志等。如果必要的话，除了要注明该类或接口应该如何使用，还需要注明不应该如何使用。

方法注释的内容通常包括该方法的用途、该方法如何工作、方法调用代码示范、必须传入什么样的参数(@param)给这个方法、异常处理(@throws)和返回值(@return)等。

2.6 本章小结

本章主要学习 Windows 平台下 Android 应用程序开发环境的搭建方法,并利用开发环境创建了第一个 Android 应用程序。学习了典型 Android 应用程序的构成、布局文件等,并对涉及的代码进行初步介绍。进一步学习 Android 应用的组成、Activity 的概念及其生命周期,以及如何编写规范的应用程序代码。

习 题

1. 简述 Android 开发环境搭建的步骤。

2. 尝试安装 Android 开发环境,并记录安装和配置过程中所遇到的问题。

3. 一个 Android 工程项目包含哪些资源文件?它们分别位于项目文件夹的什么位置?有什么作用?

4. 新建一个 Android 应用程序,打开其 AndroidManifest. xml 配置文件,了解各组成成分及其功能。

5. 假设创建 Android Application Project 时使用的包名是 com. superstar,那么程序中的 Java 源文件使用的包名是什么?Java 源文件保存在工程的哪个目录中?

6. 如果在程序中想要使用一个图像文件,应该将这个文件放置到工程的哪个目录中?

7. 将 SDK 自带的 API Demos 示例导入 Eclipse 开发环境中,通过浏览代码,了解 Android 应用程序的组成和编程风格。

8. 一个应用程序中只能有一个 Activity 对象吗?

9. 编译工程后得到的目标文件扩展名是什么?该文件被保存在工程的哪个目录中?

10. Android 应用程序由哪些部分组成?它们之间的关系是什么?

11. 简述 Android 系统的 4 种基本组件 Activity、Service、BroadcastReceiver 和 ContentProvider 的用途。

12. 简述 Activity 生命周期的 4 种状态以及状态之间的变换关系。

13. 对一些资源以及状态的操作保存,最好是在 Activity 生命周期的哪个方法中进行?

14. 如果后台的 Activity 由于某原因被系统回收了,如何在被系统回收之前保存当前状态?

第3章 Android应用程序的调试和发布

本章首先了解Android应用程序的一般开发流程，然后介绍Android应用程序的调试过程、调试工具和调试方法，以及应用程序的签名、打包和发布过程。

3.1 Android应用程序的一般开发流程

配置好Android开发环境后，应用程序一般按照以下流程完成开发。

1. 创建工程项目

在Eclipse中新建一个Android工程项目，设置Android Project名称、Package名称、应用程序名称、Activity的名称，选择Android API版本。

Android应用程序一般包含Activity和资源文件，如布局资源文件、文字资源文件等。Android应用程序就是由多个Activity间的相互交互和跳转构成的，所以Activity是应用程序必备的部分。启动应用程序时第一个运行的主Activity一般在创建工程项目时就同时创建，在其中可以指定处理逻辑、显示XML布局信息等。对于要实现多Activity跳转的情况，如菜单跳转、单击按钮后弹出另一个Activity、事件监听和捕捉用户的操作事件等的处理等，就需要设计多个Activity。Activity都要创建到src文件夹中相应的包下，并且需要将新添加的Activity类信息添加到AndroidManifest.xml配置文件中。

2. 用XML构建基本的布局和控件

开发Android应用程序，一般需要设计用户界面，可以使用XML布局文件描述应用程序界面。布局文件和资源文件一般存放在工程的res文件夹下，一般有定义界面外观的layout文件夹以及定义参数资源的values文件夹。基本的布局构件在布局文件res/layout/×××.xml文件中。一般地，main.xml或activity_main.xml描述了主Activity的布局信息。

对于界面中出现的文字，例如菜单名字、标题等，虽然可以直接写在Java文件中，但建议先写在res/values/strings.xml文件里，然后再在Java文件引用。这样处理有诸多好处，例如，以后要修改某个字符串，直接改strings.xml文件就行，否则必须在Java程序里找到所有使用这个字符串的位置修改，不仅费时费力，还容易遗漏。如果要开发多语言



版本的话，使用 strings. xml 文件定义字符串则更为方便，只需在 strings. xml 文件中修改一次，所有的界面文字就都随之改成某种语言了。另外，还可以采用 3.3 节介绍的方法处理多种语言版本的应用程序。

如果应用程序涉及数据处理方面的操作，还要设计数据存储方式，常见的数据来源包括 SharedPreferences、文件系统、数据库、ContentProcvider、网络等。此时要明确数据的格式、内容、存储方式等。

3. 编写、调试 Java 程序

在 Java 程序中实例化 XML 的布局和控件，实现业务逻辑。在开发过程中可以使用 Eclipse 提供的各种调试和测试工具，具体使用方法见本章后续介绍。

4. 修改 AndroidManifest. xml 文件

在运行程序之前必须在 AndroidManifest. xml 文件中修改应用程序的版本信息，声明程序中所有用到的 Activity、Service、Receiver 等，添加程序运行过程中需要的各种权限，如发送短信、访问网络等，否则程序发布后相关功能将无法使用。

开发过程中可以使用 Android 模拟器运行和调试程序，也可以通过数据线，直接使用安装了 Android 系统的智能移动设备运行和调试。

3.2　程序调试的常用方法和调试工具

调试是编程人员必须面对的工作。在开发 Android 应用程序时，可以直接使用 Eclipse 提供的 Java 调试器，也可以使用 ADT 插件的纠错与调试工具（如 DDMS、LogCat 等）来调试 Android 软件项目，输出错误信息。

3.2.1　使用 Eclipse 的 Java 调试器

Eclipse 提供了一个内置的 Java 调试器，提供了所有标准的调试功能，包括单步执行、设置断点和值、检查变量和值、挂起和恢复线程等。

Eclipse 调试器本身是 Eclipse 内的一个标准插件集。单击 Eclipse 窗口右上角的 Debug 按钮，可以切换到 Debug 视图，如图 3-1 所示。如果 Eclipse 窗口右上角没有 Debug 按钮，则单击 Open Perspective 按钮，在弹出的对话框中选择 Debug 选项即可，如图 3-2 所示。单击 Eclipse 窗口右上角的 Java 按钮，可以回到 Java 视图。

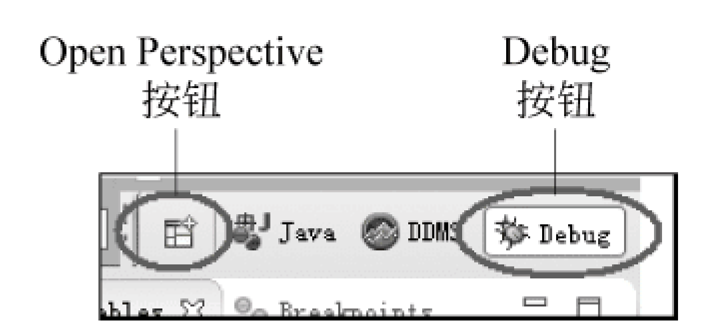


图 3-1　切换到 Debug 视图的按钮

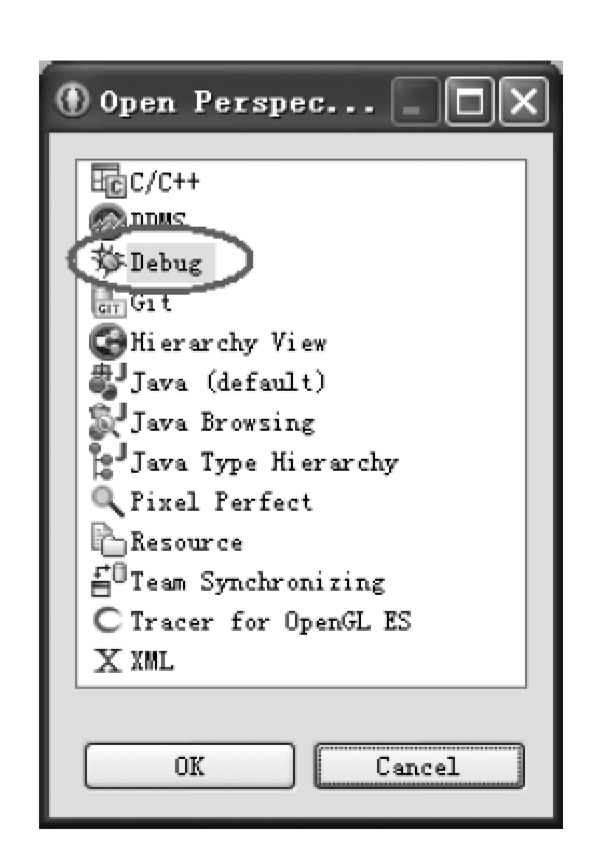


图 3-2　Open Perspective 对话框

Debug 视图的布局如图 3-3 所示。Debug 视图用于在工作台中管理程序的调试或运行。它可以显示每个调试目标中挂起线程的堆栈框架。程序中的每个线程都显示为树中的一个节点，Debug 视图显示了每个运行目标的进程。如果某个线程处于挂起状态，其堆栈框架显示为子元素。

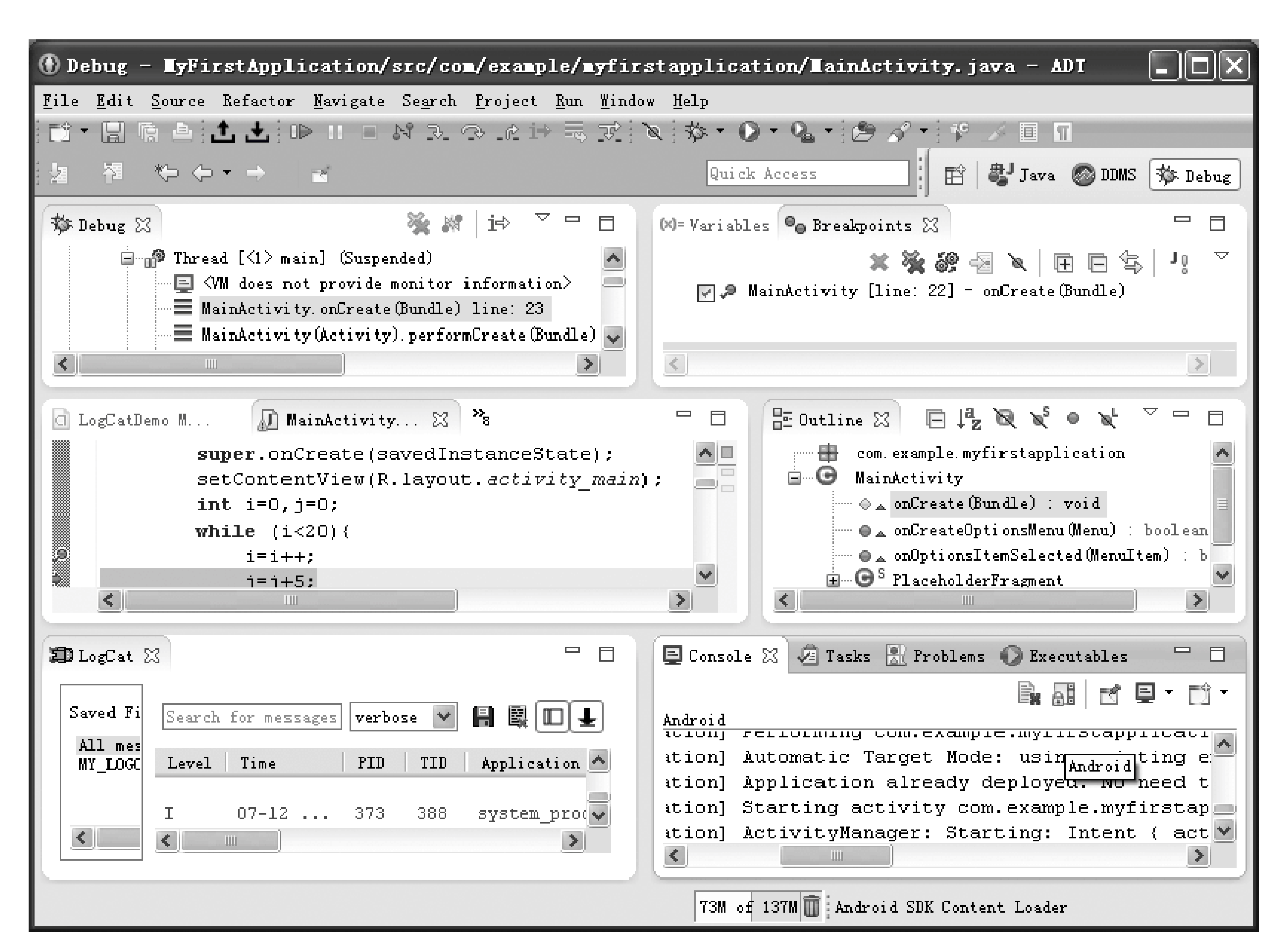


图 3-3　Debug 视图

Debug 视图中的 Debug 面板如图 3-4 所示，管理与程序调试相关的功能。面板中的视图呈树状结构，每一个线程对应一个树节点。图中显示的是暂挂线程 Main 的调试堆栈帧结构。

最常见的调试方法是设置断点，这样可以方便地检查条件语句或循环内的变量和值。

设置断点的方法：在 Java 视图的 Package Explorer 面板中双击需要设置断点的源代码文件，在右侧编辑器中打开它。将鼠标放在代码左侧的标记栏上，双击即可设置断点，如图 3-5 所示。设置断点时要注意，不要将多条语句放在一行上，不能为同一行上的多条语句设置行断点，这样也无法单步执行。

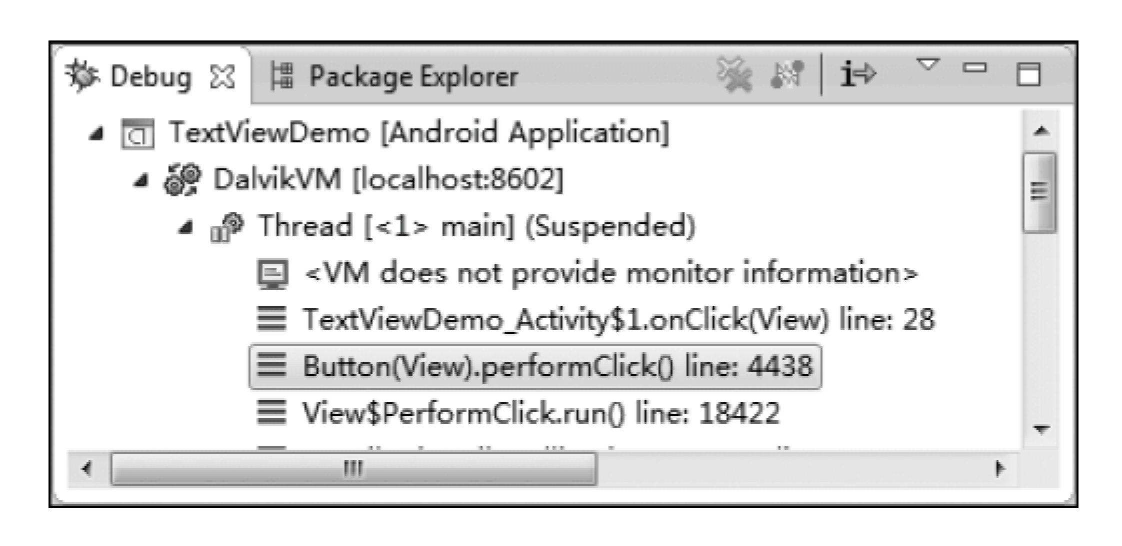


图 3-4　Debug 面板

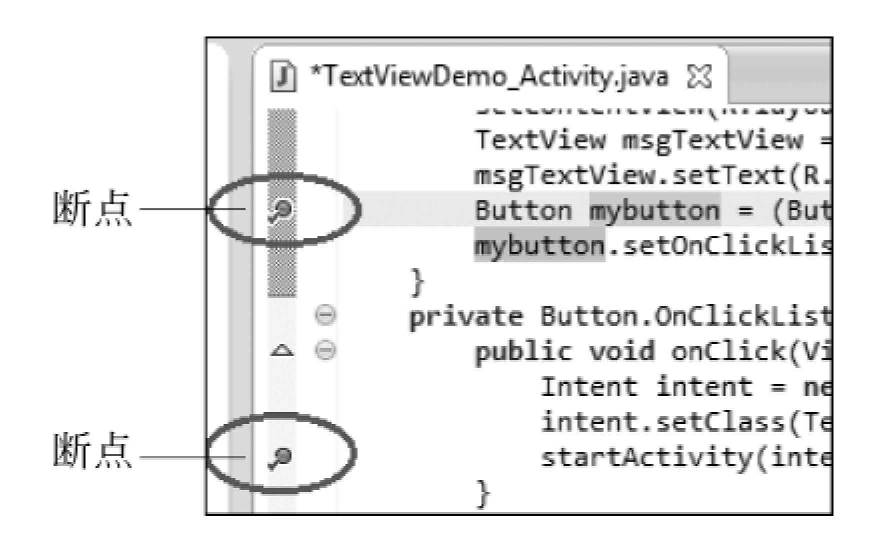


图 3-5　编辑器左侧设置的两个断点

Debug 视图中的 Breakpoints 面板列出当前工作区设置的所有断点，如图 3-6 所示。列表中显示出所有断点所属的类、断点所在的行号，以及所属的方法等。

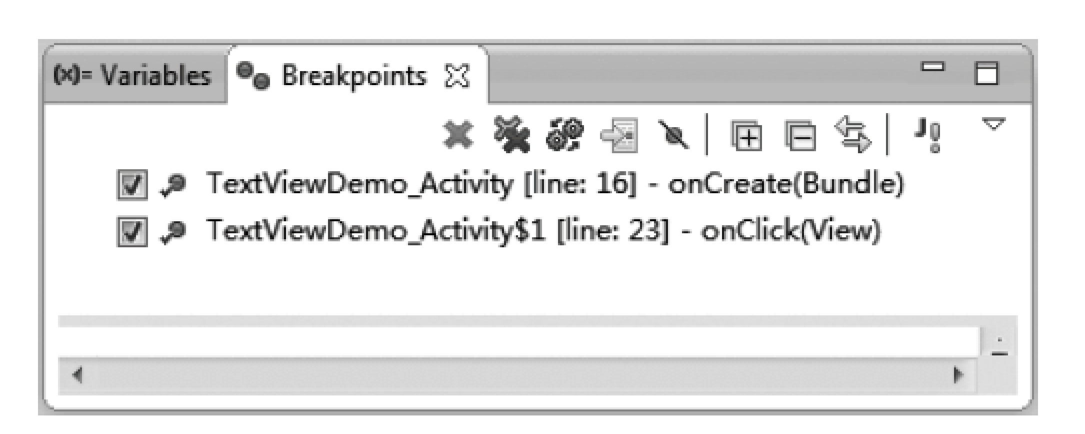


图 3-6　Breakpoints 面板

Eclipse 提供 3 种方式来启动程序的调试，分别是通过执行 Run→Debug 命令、工具栏 Debug 图标按钮、快捷键 F11。

在调试过程中，可以利用 Variables 面板随时查看断点时的变量值，如图 3-7 所示。Variable 面板显示了选中的堆栈帧中的变量值。要查看所请求的变量，只需展开 Variables 视图中的树直到所请求的元素为止。

当调试器停止在一个断点处时，可以从 Debug 视图工具栏中单击 Step Over 按钮，如图 3-8 所示，继续单步执行代码。

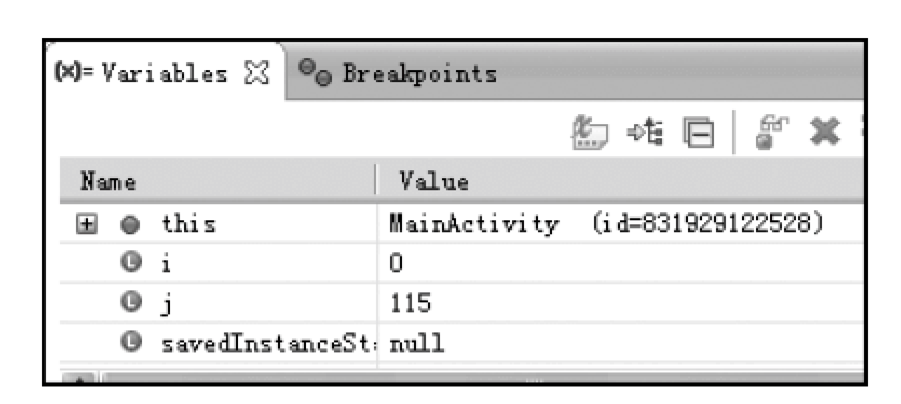


图 3-7　Variables 面板



图 3-8　工具栏中的 Step Over 按钮

Eclipse 提供了 Step Into、Step Over、Step Return 3 个命令来支持单步调试。三者的具体区别：Step Into(快捷键 F5)在单步执行过程中，遇到子函数就进入并且继续单步执行；Step Over(快捷键 F6)是在单步执行过程中，在函数内遇到子函数时不会进入子函数

内单步执行，而是将子函数整个执行完再停止，也就是把子函数整个作为一步；单步执行到子函数内时，用 Step Return(快捷键 F7)命令就可以一步执行完子函数余下部分，并返回到上一层函数

通过执行 Run→Disconnect 命令、或单击工具栏 Disconnect 图标按钮都可以终止程序的调试。

3.2.2 图形化调试工具 DDMS

DDMS(Dalvik Debug Monitor Service)主要用于监控 Android 应用程序的运行并打印日志、模拟电话打入与接听、模拟短信收发、虚拟地理位置等。DDMS 集成在虚拟机 Dalvik 中，主要用于管理运行在模拟器或设备上的进程，并协助用户进行调试。可以用它来处理进程、选择特定应用程序调试、生成跟踪数据、查看堆和线程数据、对模拟器或设备进行屏幕快照等。

单击 Eclipse 窗口右上角的 DDMS 按钮，就可以切换到 DDMS 视图。如果右上角没有 DDMS 按钮，可以单击 Open Perspective 按钮，在弹出的对话框中选择列表中的 DDMS 项即可。

DDMS 视图如图 3-9 所示。DDMS 视图中的左上部是 Devices 面板，在这里可以看到与 DDMS 连接的设备终端的信息及设备终端上运行的应用程序。如果没有终端设备在运行，则这个面板中的内容为空。在 Devices 面板中，可以设置应用程序更新 Heap 状态、更新 Thread 状态，或者直接停止某个应用程序的执行。同时，Devices 面板中还可以截取手机屏幕。

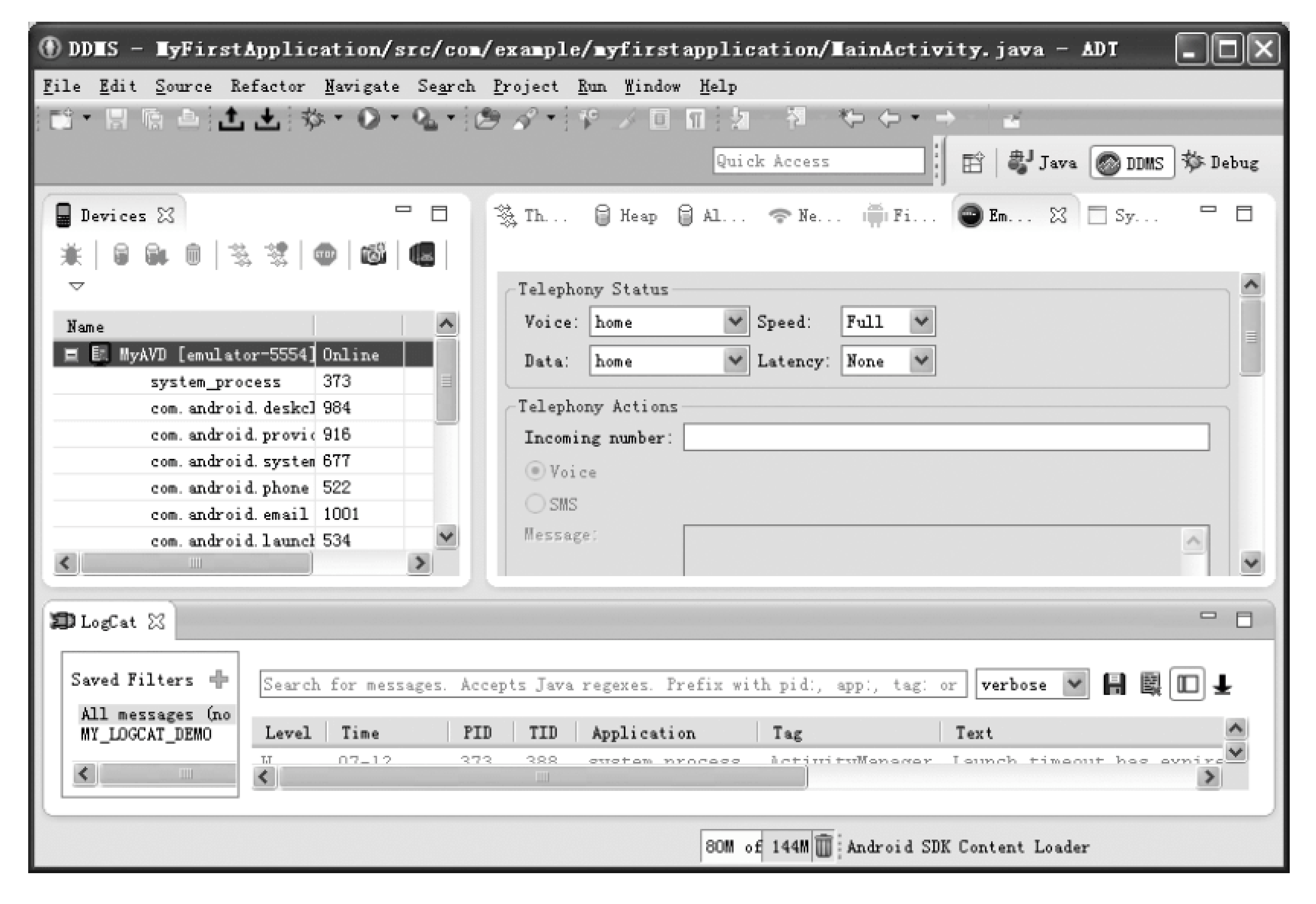


图 3-9　DDMS 视图

如果支持 GSM 等通信网络的话，在 DDMS 中的 Emulator Control 面板中可以向模拟器 AVD 中打入电话或发送短信，还可以虚拟模拟器的位置信息等。

DDMS 有如下各种输出面板，可用于获取程序调试过程中的各种信息。

(1) Thread 更新信息：要使该窗口输出信息，需要单击 Devices 面板中的 Update Threads 按钮。这个窗口主要显示应用程序当前状态下所有正在执行的线程的状态。

(2) Heap 更新信息：要使该窗口输出信息，需要单击 Devices 面板中的 Update Heap 按钮。这个窗口主要显示当前状态下堆分配与回收信息。

(3) File Explorer：该窗口主要显示 Android 模拟器中的文件，如果模拟器启动时加载了 SD 卡，也可以在该窗口中查看 SD 卡的信息。

(4) LogCat：显示应用程序的运行信息、调试信息、警告信息、错误信息等。不同类型的信息文字具有不用的颜色。当 LogCat 输出的信息量很大时可以根据需要对其内容进行过滤。LogCat 的具体使用方法见 3.2.3 节。

3.2.3　查看工程项目在运行过程中的日志信息

在调试程序的过程中，经常需要察看程序运行过程中的状态信息，可以利用 LogCat 工具输出这些信息。

LogCat 是 Android 系统提供的一个调试工具，用来获取系统日志信息。LogCat 能够捕获的信息包括 Dalvik 虚拟机产生的信息、进程信息、ActivityManager 信息、PackagerManager 信息、Homeloader 信息、WindowsManager 信息、Android 运行时信息和应用程序信息等。

LogCat 可以显示在 Eclipse 集成开发环境中。打开 LogCat 面板的方法：执行 Window→Show View→Other 命令，打开 Show View 的选择菜单，然后在 Andoird→LogCat 中选择 LogCat。LogCat 面板打开后，便显示在 Eclipse 的下方区域，如图 3-10 所示。

图 3-10　LogCat 面板及其显示的信息

LogCat 面板的右上方有一个下拉列表，前 5 个选项分别是 verbose、debug、info、warn、error，它们分别表示 5 种不同类型的日志信息，分别是详细(Verbose)信息、调试(Debug)信息、通告(Info)信息、警告(Warn)信息、错误(Error)信息，它们的级别依次增

高。单击这些选项，可以使 LogCat 面板中仅输出指定类型的日志信息，级别高于所选类型的信息也会在 LogCat 中显示，但级别低于所选类型的信息则不会显示。

LogCat 提供了“过滤”功能。单击面板左侧的＋按钮，则弹出如图 3-11 所示的对话框，可以添加一个新的过滤器。用户可以根据日志信息的标签（Tag）、产生日志的进程编号（PID）或信息等级（Level）等对显示的日志内容进行过滤。

图 3-11　设置过滤器对话框

利用 LogCat，可以在程序中预先设置一些日志信息，当程序运行时，这些日志信息就会输出到 LogCat 窗口。这样，就可以在调试程序的过程中通过 LogCat 查看工程项目在运行过程中的状态。具体方法如下。

步骤 1：在程序中使用 import 语句引入 android. util. Log 包文件。

步骤 2：调用 Log. v()、Log. d()、Log. i()、Log. w()、Log. e()方法在程序中设置“日志点”。

Log. v()用来输出详细信息，Log. d()用来输出调试信息，Log. i()用来输出通告信息，Log. w()用来输出警告信息，Log. e()用来输出错误信息。这些函数都有两个参数，两个参数的数据类型都是字符串。第一个参数是日志的标签（Tag），第二个参数是在 Logcat 中要显示的日志内容。标签是一个字符串，通常在程序中将其定义成符号常量。标签可以帮助人们在 LogCat 中找到目标程序生成的日志信息，同时也能够利用标签对日志信息进行过滤。

步骤 3：当程序运行到“日志点”时，预先设置的日志信息便被发送到 LogCat 窗口中。

在调试程序时可以用这种方法显示日志信息，然后判断“日志点”信息与预期的内容是否一致。进而判断程序是否存在错误。

【例 3-1】　工程 03_LogCatExample 演示了 Log 类的具体使用方法。

MainActivity 类的代码如下：

```
package edu.hebust.zxm.logcatexample;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;                          //引入 android.util.Log

public class MainActivity extends Activity {
    final static String TAG= "MY_LOGCAT_EXAMPLE";   //定义一个用于日志标签的符号常量
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.activity_main);
            Log.v(TAG,"My information:Verbose");          //产生一个详细信息
            Log.d(TAG,"My information:Debug");            //产生一个调试信息
            Log.i(TAG,"My information:Info");             //产生一个通告信息
            Log.w(TAG,"My information:Warn");             //产生一个警告信息
            Log.e(TAG,"My information:Error");            //产生一个错误信息
        }
    }
```

上例 LogCatExample 工程的运行结果如图 3-12 所示，LogCat 对不同类型的信息使用了不同的颜色加以区别。

Declaration　Console　LogCat

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to l　verbose

L...	Time	PID	TID	Application	Tag	Text
I	09-...	384	398	system_pro...	Choreographer	Skipped 64 frames! The applicatio d.
V	09-...	1687	1687	edu.hebust...	MY_LOGCAT_EXAMPLE	My information: Verbose
D	09-...	1687	1687	edu.hebust...	MY_LOGCAT_EXAMPLE	My information: Debug
I	09-...	1687	1687	edu.hebust...	MY_LOGCAT_EXAMPLE	My information: Info
W	09-...	1687	1687	edu.hebust...	MY_LOGCAT_EXAMPLE	My information: Warn
E	09-...	1687	1687	edu.hebust...	MY_LOGCAT_EXAMPLE	My information: Error
D	09-...	1687	1687	edu.hebust...	gralloc_goldfish	Emulator without GPU emulation det
I	09-...	384	398	system_pro...	ActivityManager	Displayed edu.hebust.zxm.logcatexa
I	09-...	384	398	system_pro...	Choreographer	Skipped 78 frames! The applicatio d.

图 3-12　LogCatExample 工程输出的 LogCat 信息

当 LogCat 输出的信息很多时，可以添加过滤器，使其仅显示需要或感兴趣的信息。过滤器可以根据标签的内容进行过滤，所以使用过滤器之前要为 Log. v()、Log. d()、Log. i()、Log. w()、Log. e()方法先定义一些标签。

具体过滤方法：单击面板右上角的＋按钮，填入过滤器的名称——LogcatFilter，设置过滤条件为"标签＝MY_LOGCAT_EXAMPLE"。LogCat 过滤后的输出结果如图 3-13 所示，无论什么类型的日志信息，属于哪一个进程，只要标签为 MY_LOGCAT_EXAMPLE，都将显示在 LogCat 面板中。

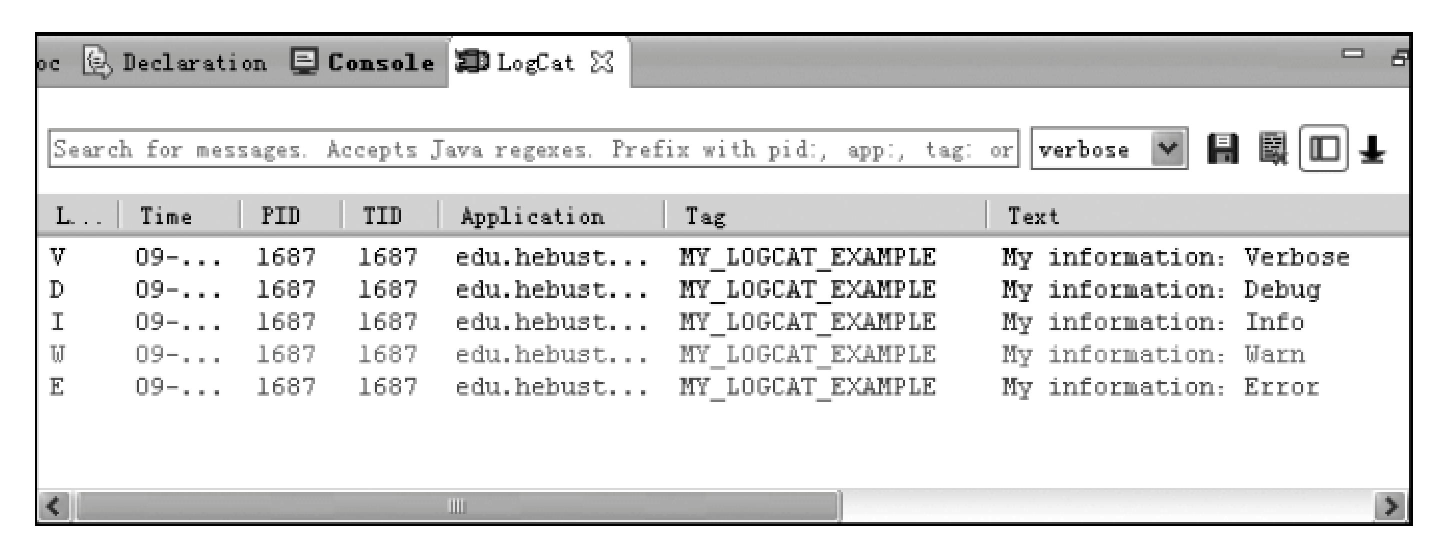


图 3-13　过滤后的 LogCat 信息

3.2.4　Dev Tools

Android 系统提供了用于调试和测试的工具集 Dev Tools，包括 Developer options、

Package Browser、Pointer Location、Running processes 等一系列各种用途的小工具。在应用列表中单击 Dev Tools，就可以进入到 Dev Tools 的使用界面，如图 3-14 所示。



图 3-14　测试工具集 Dev Tools

1. 开发者选项小工具

开发者选项(Developer options)小工具中包含了程序调试的相关选项，单击功能后面的选择框，出现勾选符号时表示功能启用，如图 3-15 所示。设置项发生变化后，模拟器会自动保存设置。



图 3-15　Developer options 小工具

开发者选项有以下常用的选项(图 3-15 中没显全，所以看不到)。

(1) 选择调试应用：为“等待调试器”选项指定应用程序，如果不指定，选项为 none，“等待调试器”选项将适用于所有应用程序。该选项可以有效防止 Android 程序长时间停留在断点而产生异常。

(2) 等待调试器：阻塞加载应用程序，直到关联到调试器(Debugger)。用于在 Activity 的 onCreate()函数的断点调试。

(3) 显示面(surface)更新：选中该选项时，界面上任何被重绘的矩形区域会闪现粉红色，有利于发现界面中不必要的重绘区域。

(4) 显示 CPU 使用情况：在屏幕顶端显示 CPU 的使用率，包括总的 CPU 使用率和当前进程的 CPU 使用率。

(5) 不保留活动：Activity 进入停止状态后立即销毁，用于测试在方法 onSaveInstanceState()、onRestoreInstanceState()和 onCreate()中的代码。

2. Package Browser 小工具

Package Browser 是 Android 系统中的程序包查看工具，能够详细显示已经安装到 Android 系统中的程序信息，包括包名称、应用程序名称、图标、进程、用户 ID、版本、APK

文件保存位置和数据文件保存位置等。Package Browser 的界面如图 3-16 所示。进一步单击应用程序，可以查看该程序所包含 Activity、Service、BroadcastReceiver 和 Provider 的详细信息。例如，Package Browser 查看 Calculator 程序的相关信息如图 3-17 所示。



图 3-16　Package Browser 小工具

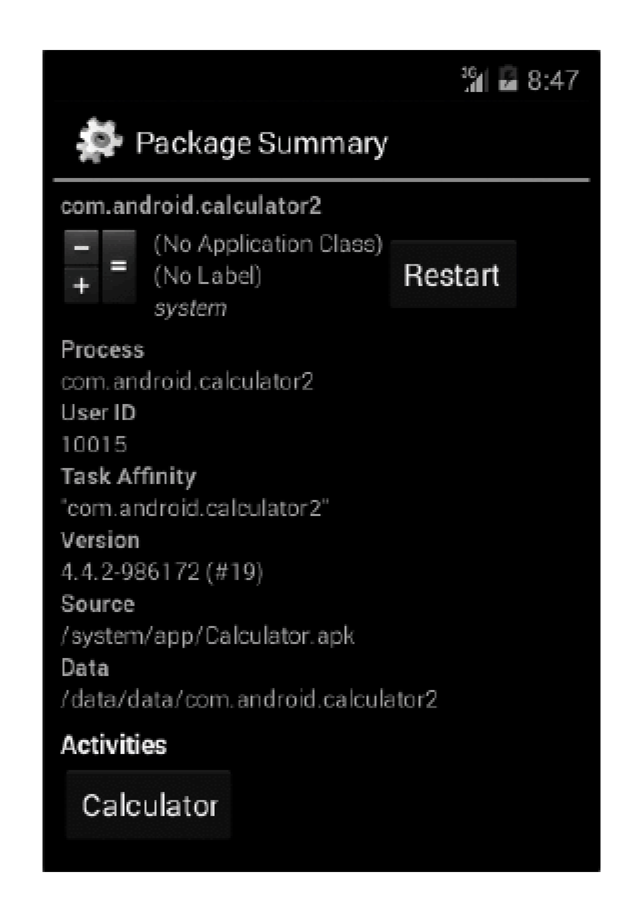


图 3-17　Calculator 程序的信息

3. Running processes 小工具

Running processes 能够查看在 Android 系统中正在运行的进程，并能查看进程的详细信息，包括进程名称和进程所调用的程序包。Running processes 的界面如图 3-18 所示，进程(例如 com. android. calendar)的详细信息如图 3-19 所示。

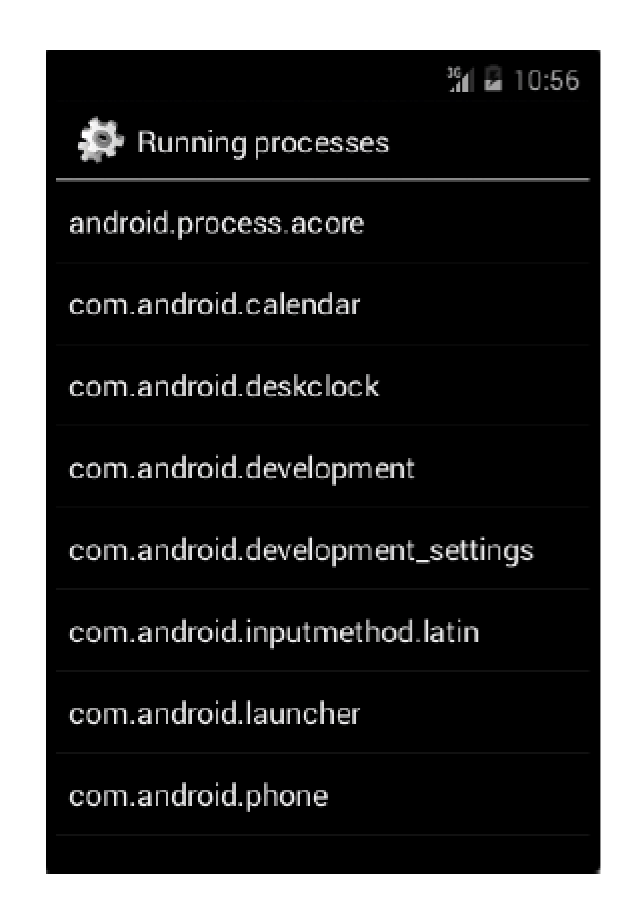


图 3-18　Running processes 小工具

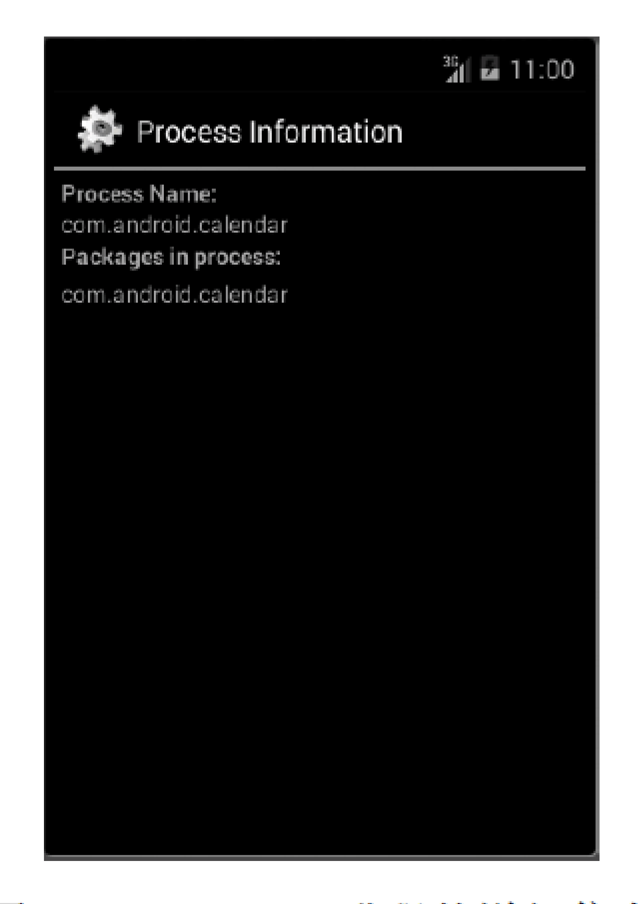


图 3-19　calendar 进程的详细信息

3.3　应用程序的国际化

Android 应用程序可以方便地实现不同语言和地区的国际化。当应用程序运行时，会根据系统的地区和语言设置自动选择界面中显示的语言。

要想使自己的应用程序实现国际化，需要对应用程序和 res 资源文件做相应的配置。首先，需要创建带有语言代号和地区代号的资源文件夹，方法是在原有文件夹的名称后面添加“-语言代号-r 地区代号”。常用的语言代号有 zh、en、fr 等，分别表示中文、英文、法文；常用的地区代号有 US 和 UK，表示美国和英国。要注意的是，地区代号前面的 r 是必需的。例如，原资源文件夹的名称为/values，则新文件夹的名称为/values-zh-rCN。

需要自动适配多少个国家和地区，就需要创建多少个语言代号和地区代号不同但其他名称相同的文件夹。在这些文件夹中分别放置名称相同，但内容不同的资源，如字符串、图片等。

程序中需要国际化的部分必须使用资源 ID 引用资源文件夹中的字符串或图片。应用程序在运行时会按照匹配规则到资源文件夹中引用资源，首先会找语言、地区完全匹配的资源，如果没有地区匹配的，则查找语言匹配的资源，如果没有语言匹配的则按照缺省的方式使用默认的资源。

【例 3-2】 示例工程 03_InternationalExample，演示了一个国际化的程序。

步骤 1：首先在工程目录中的 res 文件夹下创建新文件夹，分别是 values-zh-rCN、values-en-rUS、drawable-zh-rCN 和 drawable-en-rUS。

在前两个文件夹中分别创建文件 strings. xml；在后两个文件夹中分别放置图片 flag. jpg，这两个图片的文件名相同，但内容不同，分别为中国和美国的国旗图案。

res/values-zh-rCN/strings. xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">欢迎!</string>
    <string name="local">地区:中国</string>
</resources>
```

res/values-en-rUS/strings. xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Welcome!</string>
    <string name="local">Local:USA</string>
</resources>
```

注意上述两个 strings. xml 文件中，定义了相同名字的字符串资源，但其资源值使用的语言不同。

步骤 2：定义 XML 布局文件。布局中引用了资源文件中的字符串和图片，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```




```
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
                <!--引用了资源文件 strings.xml 中的字符串:hello-->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/local" />
                <!--引用了资源文件 strings.xml 中的字符串:local-->
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/flag"/>
                <!--引用了 res/drawable 文件夹中的资源文件 flag.jpg-->
</LinearLayout>
```

步骤 3：定义 MainActivity 类文件，实例化布局，其内容如下：

```
package edu.hebust.zxm.internationalexample;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.setTitle("地区和语言的国际化示例");
    }
}
```

当 Android 系统的语言设置为“中文(简体)”时，其运行结果如图 3-20(a)所示；当系统的语言设置为“English(United States)”时，其运行结果如图 3-20(b)所示。程序不需要做任何修改，就可以根据系统的地区和语言设置自动选择界面中显示的语言。

需要注意的是，本例中的 Activity 标题在任何语言环境下都显示中文“地区和语言的国际化示例”，这是因为设置该标题是使用下面的语句实现的。

```
this.setTitle("地区和语言的国际化示例");
```

该语句没有采用资源 ID 的方式从资源文件中引用字符串，而是使用了字符串常量，所以程序中的显示不会根据系统的语言设置而自动改变。



(a)



(b)

图 3-20 示例程序的运行结果

3.4 应用程序的发布

3.4.1 程序发布前的收尾工作

1. 使用图片设置应用程序图标

Android 标准图标根据手机分辨率的不同，分别有 36×36、48×48/、72×72 3 种尺寸，分别放置在 res 文件夹中的 drawable-ldpi/、drawable-mdpi、drawable-hdpi 文件夹中。不同的目录用来存储不同尺寸的图标，在 AndroidManifest.xml 中只需要用“@drawable/图标文件名”调用就可以，系统会根据移动设备的屏幕分辨率去使用不同目录下的图标文件。

图标文件一般使用 PNG 或 JPG 格式的图片，可以按照上述分辨率准备 3 个图标文件，分别复制到 res/drawable-ldpi、res/drawable-mdpi、res/drawable-hdpi 文件夹中。设置了自定义图标的应用程序，如图 3-21 所示。



图 3-21 使用自定义的图标

2. 添加最终用户许可协议

用户许可协议用于保护作者的知识产权。当应用程序第一次启动时，会显示预先设置好的用户许可协议文字，用户选择“接受”或“拒绝”。只有用户选择“接受”，程序才会继续运行，否则会退出。

一般将用户许可协议的内容存放在一个文本文件中。设计一个 EULA 类，为该类设计以下几个方法：show()方法，用于显示用户许可协议窗口和处理用户的选择；accept()方法，当用户选择“接受”时，进入运行应用程序的状态；refuse()方法，当用户选择“拒绝”时，停止 Activity 的运行；readEula()方法，从文本文件中读取许可协议的内容。具体步骤可以参考网址 http://bees4honey.com/blog/tutorial/adding-eula-to-android-app/。

3. 程序的版本管理

修改 AndroidManifest.xml 文件中应用程序的版本信息。

3.4.2　APK 文件的签名和打包

当完成 Android 应用程序的开发后，虽然可以使用模拟器显示其在真实手机上的运行效果，但模拟器毕竟还不是真正的手机，应用程序也并未真正发送到真实的手机上。和其他 Java 应用程序不同，Android 应用程序一般要打包成 APK 文件后再发送到真实的手机上。将 APK 文件直接传入 Android 模拟器或 Android 手机中即可安装。

APK 文件中包含了与某个 Android 应用程序相关的所有文件，如 AndroidManifest.xml、应用程序代码(.dex 文件)、资源文件等。一个工程只能打包进一个 APK 文件中。

在 Android 平台上开发的所有应用程序，在安装到模拟器或手机前都必须进行数字签名。如果强行将没有数字签名的 Android 程序安装到模拟器中，将会返回错误提示。Eclipse 开发环境中，在将程序安装到模拟器之前，ADT 会利用内置的 debug key 为 APK 文件自动进行数字签名，这使编程者可以快速完成程序的调试。

如果想将其上传到 Android 电子市场上供别人下载，则不能使用 debug key，而必须使用私有密匙对 Android 程序进行数字签名。Android 电子市场一般要求发布的应用程序是经过签名的且不能是 Debug 模式下的签名。另外要特别注意，同一个应用的不同版本，一定要使用同一个签名，这样安装程序的时候，才会自动升级，用新版本代替旧版本。否则，签名不同时系统会认为是不同的应用。

签名可以采用命令行方式，也可以在 Eclipse 中使用 ADT 完成。本节以在 Eclipse 中使用 ADT 完成签名为例介绍签名和打包的方法。

步骤 1：在 Eclipse 中右击某个拟生成 APK 的应用程序，执行 Android Tools→Export Signed Application Package 命令。

步骤 2：选择欲导出的工程项目名，之后单击 Next 按钮，弹出如图 3-22 所示的对话框。选择 Create new keystore 单选按钮，选择 Keystore 位置，即存放该签名文件的地址，输入自己的密码。单击 Next 按钮。

图 3-22　Keystore 设置

如果在上述对话框中选择 Use existing keystore 单选按钮，则可以选择以前生成的 APK 文件，在输入相应的密码后会弹出对话框，可以再次对其生成签名。

步骤 3：输入相应的签名信息，如图 3-23 所示。根据具体情况填写后，单击 Next 按钮。

Export Android Application
Key Creation
Alias: MyKey
Password: ******
Confirm: ******
Validity (years): 25
First and Last Name: Xuemei Zhang
Organizational Unit:
Organization: hebust
City or Locality: Shijiazhuang
State or Province: Hebei
Country Code (XX): CN
< Back | Next > | Finish | Cancel

图 3-23 设置签名信息

步骤 4：设定具体的 APK 文件存储路径，如图 3-24 所示。单击 Finish 按钮生成 APK 文件并同时生成签名文件。

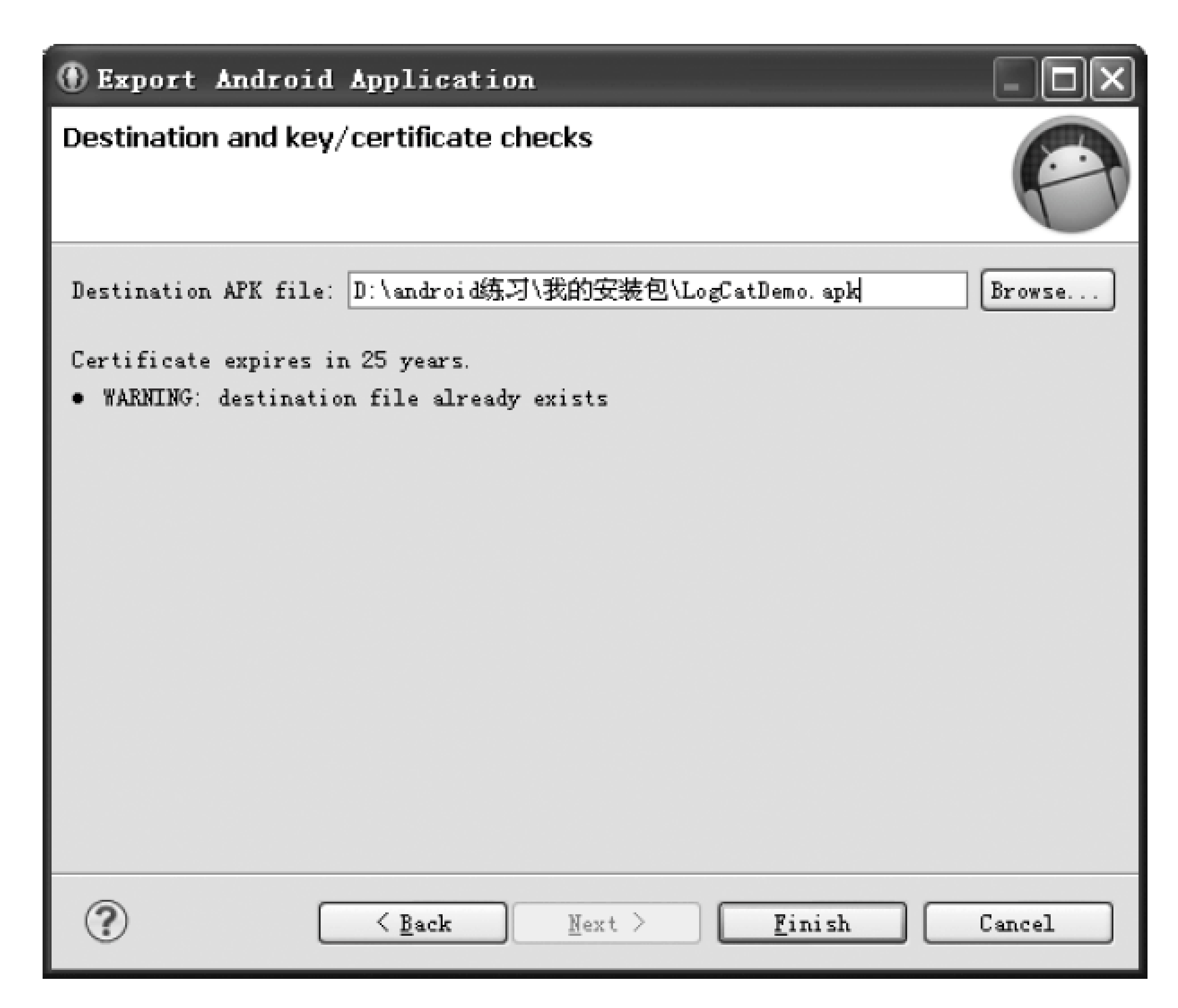


图 3-24 设置 APK 文件存储路径

除了上面提到的方法外，也可以通过 AndroidManifest. xml 来导出 APK 程序。打开相应的应用程序中 AndroidManifest. xml，单击 Manifest 标签页，再单击标签页下方的 Exporting 栏中的 Use the Export Wizard to export and sign APK，便可打开导出向导，将该项目的 APK 文件导出。与前述的过程类似，按照向导提示，在设定相应的信息后将应用程序打包为 APK 文件。

打包后的文件中包括资源文件、配置文件和可执行文件。可以使用 WinRAR 解压软件将其解压缩，会看到相应的 AndroidManifest. xml、resources. arsc 资源文件与 res 资源目录，以及一个 classes. dex 文件，如图 3-25 所示。dex 是 Dalvik VM Executes 的简称，即 Android Dalvik 可执行程序。



图 3-25　APK 文件的内容

3.4.3　APK 文件的安装

生成 APK 文件之后，需要安装到移动设备中，程序才能运行。下面的安装以 Windows 系统下 APK 文件的安装为例进行说明。

在向移动设备中安装之前，需要对其进行设置。以手机为例，首先，执行"设置"→"开发者选项"命令，勾选"USB 调试"复选框，如图 3-26 所示，完成手机设置。

APK 文件安装方法有以下 3 种。

方法 1：使用数据线将移动设备与计算机连接，然后将 APK 文件复制到移动设备的 SD 卡中，然后再进入移动设备的文件管理器，找到 APK 文件，就可以打开文件运行安装。

方法 2：使用一些客户端软件完成 APK 文件的安装，如"91 手机助手"、"360 手机助手"等。这些客户端软件都可以完成数据复制、应用程序安装与卸载、手机屏幕截图功能。在计算机中安装并运行前述客户端软件后，



图 3-26　勾选"USB 调试"复选框

APK 文件就会被自动识别，然后将移动设备连接计算机，打开手机 USB 调试模式，之后直接在计算机中打开 APK 文件，即可安装到手机。

很多品牌的移动设备都会提供与设备连接的客户端软件，提供数据复制、应用程序安装等功能，图 3-27 所示为 HTC 手机的客户端软件。运行客户端软件的应用程序安装功能，选择相应的 APK 文件，确定后就会开始安装。之后，在移动设备中就可以看到并运行相应的应用程序了。



图 3-27　移动设备专用的客户端软件

方法 3：当移动设备和计算机处于连接状态时，在 Eclipse 中运行应用程序时将不再打开模拟器，而是直接在移动设备中运行。如果已经启动了模拟器，系统会弹出对话框要求用户选择运行程序的设备，如图 3-28 所示，选择与计算机连接的移动设备即可完成程序的安装和运行。

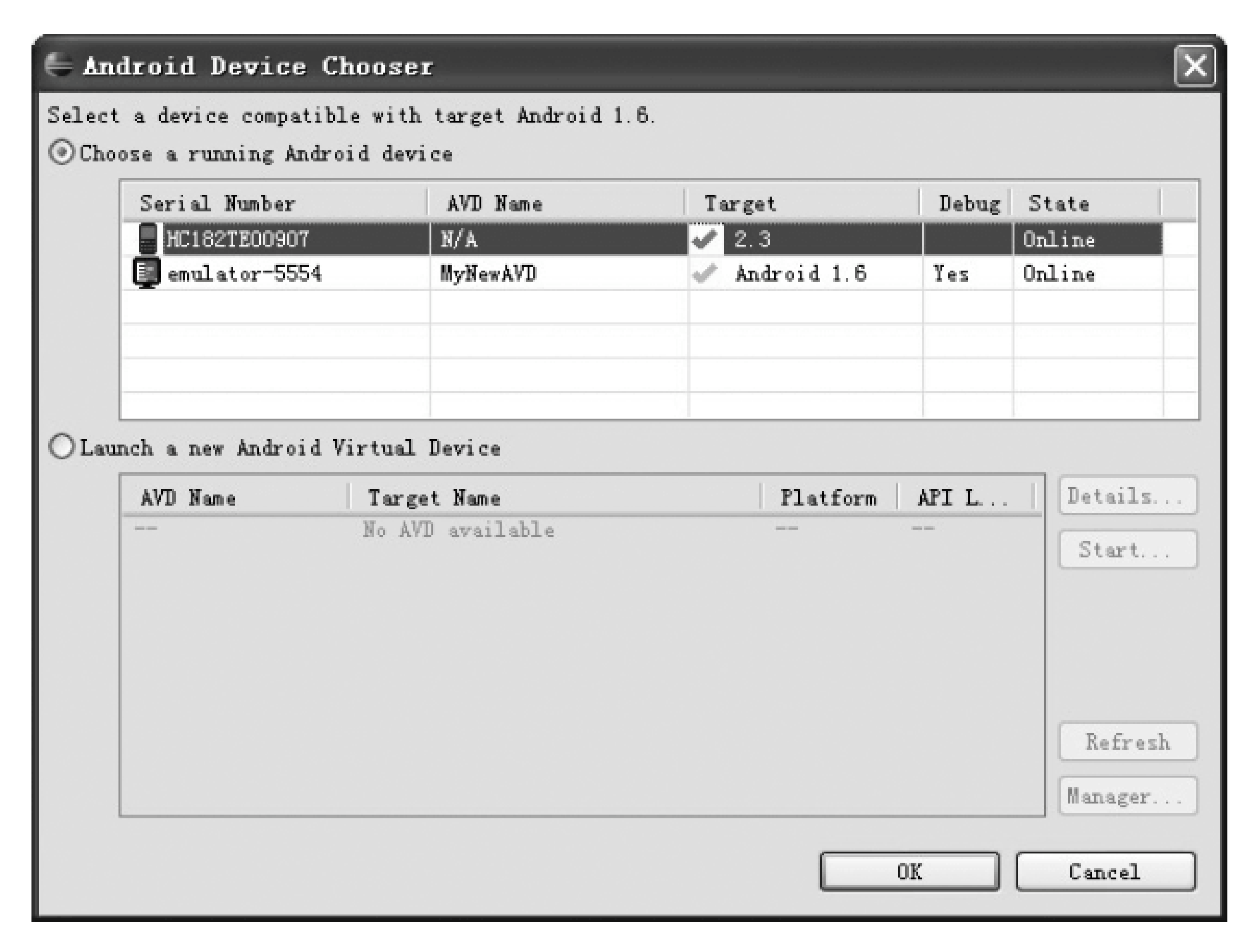


图 3-28　选择直接在移动设备上运行



3.4.4 在 Android 电子市场上发布自己的应用程序

虽然直接把 APK 文件复制给别人就可以用了，但电子市场是当前流行并且传播最快的一种软件发布途径。

Android 电子市场是一个开放的平台，提供众多 Android 平台的应用程序，用户可以按照分类查看并可直接下载、安装、评价。同时开发者可以上传和销售自己的作品，随时查看其被下载、安装的次数、星级评价等。目前有很多 Android 电子市场，其中最著名的是 Google 公司推出 Android Market，现已更名为 Google Play Store（https://play.google.com/store）。国内也有许多这样的电子市场，例如安卓市场（http://apk.hiapk.com/）、安智市场（http://www.anzhi.com/）、搜狗市场（http://app.sogou.com/）、N 多网（http://www.nduoa.com/）等，中国移动、中国电信、中国联通也都有自己的 Android 应用市场。

软件开发者在 Android 电子市场发布应用程序，一般需要注册并支付一定的费用，之后才可以在电子市场上销售他们的软件作品。通常开发者、Android 电子市场、运营商、广告商会按照协议从每次交易中分获利润。

在发布应用程序之前，要对应用程序进行签名和打包，得到一个属于应用程序的 keystore 文件和一个签过名的.apk 安装文件。另外还需要准备一些应用程序的界面截图，这些图片将会在 Android 市场里展示该应用时使用。还要为应用提供一个标题和一个简短描述。当用户在 Android 市场里搜索时，会使用到这些词汇。有时还需要为应用程序留下联系信息，其中的一项是 URL。建议这个 URL 应该是指向与应用程序有关的 Web 网页，这个网页上可以提供关于应用的更详细的介绍说明，以及其他更丰富的屏幕截图，这些信息对那些在 Android 交易市场里搜索到这个应用程序，并想了解更详细信息的人来说是非常有用的。

应用程序开发完成，并做好上述准备工作之后就可以将其发布到 Android 电子市场中，供所有人下载和使用。

3.5 本章小结

本章主要学习了 Android 应用程序开发的一般步骤，以及在 Eclipse 环境中常用的一些纠错与调试工具，包括 Eclipse 的 Java 调试器、DDMS、LogCat、Dev Tools 小工具等。并学习了对 APK 文件签名、打包、发布到应用市场的方法。

习 题

1. 简述 Android 应用程序的一般开发流程。
2. 如何在程序代码中设置断点？设置断点有什么作用？
3. 如何单步执行程序？
4. 简述图形化调试工具 DDMS 的用途。

5. DDMS 中 Log 信息分为哪几个级别?

6. 如何利用 LogCat 在程序中设置日志信息?如何对输出的日志信息进行过滤?

7. 设计一个应用程序,利用 LogCat 的输出信息测试 Activity 生命周期中各回调方法的调用条件和调用顺序。

8. 在 Android 电子市场中发布自己的应用程序需要做哪些准备?要注意什么问题?

9. 简述 APK 文件的签名和打包过程。

第4章 用户界面设计基础

本章介绍设计 Android 用户界面的基础知识，主要内容包括 XML 布局文件的设计方法、常见的界面布局方式、Widget 控件、在 Activity 中引用布局和 Widget 控件的方法，以及 Android 中的事件处理机制。

4.1 界面布局及其加载

Android 应用程序的布局可以采用 XML 布局文件来指定，也可以采用在 Activity 中书写 Java 代码的方式来设定布局，本章介绍采用 XML 布局文件的设计方法。Android 中常用的布局包括线性布局、表格布局、相对布局、框架布局和绝对布局。

4.1.1 View 类和 ViewGroup 类

在一个 Android 应用程序中，用户界面通过 View 和 ViewGroup 对象构建。View 对象是 Android 平台上表示用户界面的基本单元，一个 View 占据屏幕上的一块方形区域，存储了该特定区域的布局参数和内容。Android 平台下 View 类是所有可视化控件的基类，主要提供控件绘制和事件处理的方法。Android 中有很多种 View 和 ViewGroup，它们都继承自 View 类。创建用户界面所使用的控件都继承自 View 类，如本章介绍的 TextView、Button 和 CheckBox 等。

ViewGroup 类是 View 类的子类，与 View 类不同的是，它可以充当其他控件的容器。其主要功能是装载和管理一组 View 和其他的 ViewGroup，可以嵌套 ViewGroup 和 View 对象，而它们共同组建的顶层 View 可以由应用程序中的 Activity 中使用 setContentView()方法来显示。ViewGroup 的子类既可以是普通的 View，也可以是 ViewGroup。Android 中的一些高级控件如 Galley、GridView 等都继承自 ViewGroup。在 Android 中，为每种不同的布局提供了一个 ViewGroup 的子类，常用的布局管理器的类结构如图 4-1 所示。

用户界面 UI 一般由一组继承自 View 基类的某个可视化控件和容器 ViewGroup 构成。

View 及其子类的相关属性，既可以在布局 XML 文件中进行设置，也可以通过成员方法在代码中动态设置。任何继承自 View 的子类都拥有 View 类的属性及对应方法。

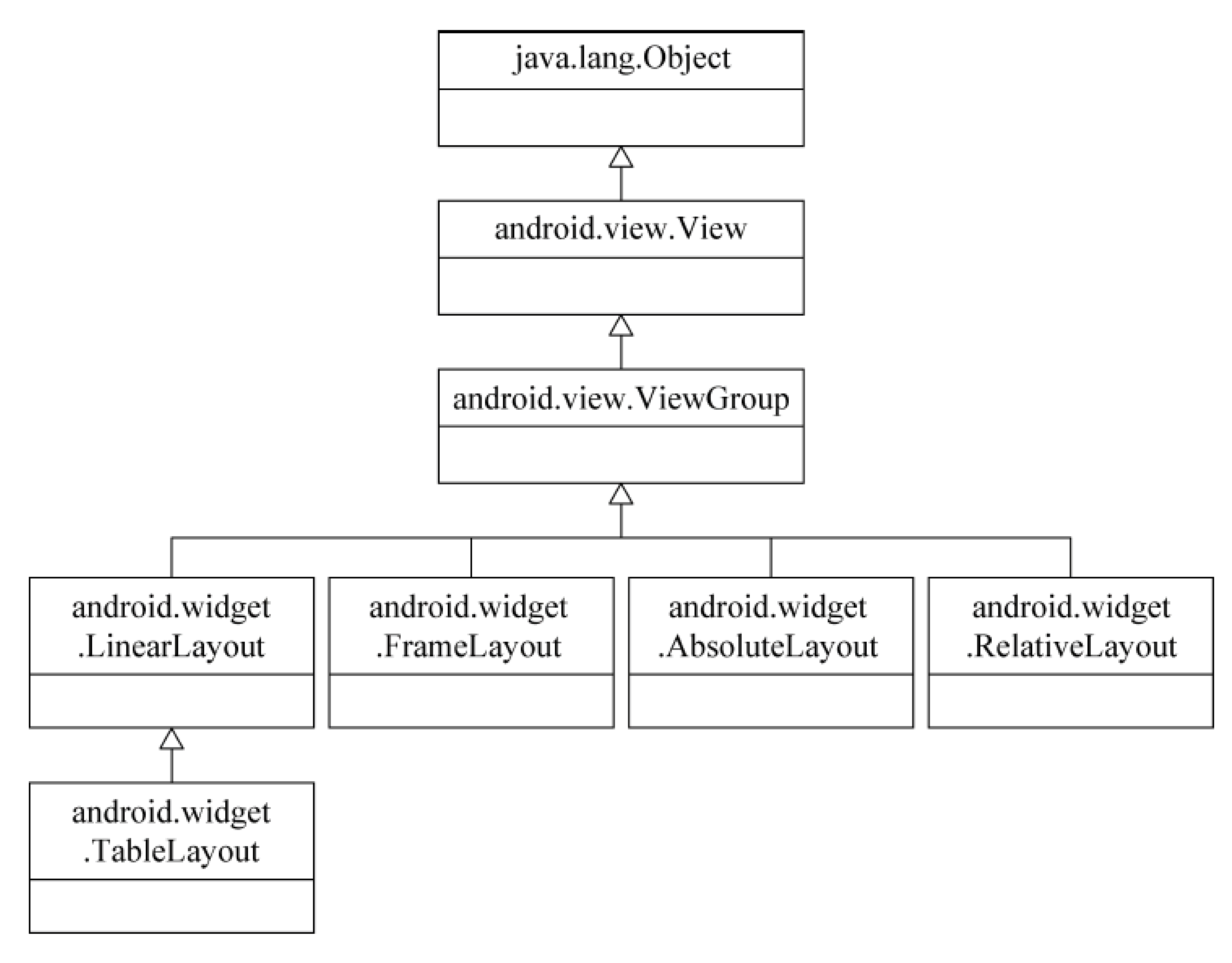


图 4-1 布局管理器的类结构

4.1.2 布局管理

Android 中的布局方式主要有线性布局(Linear Layout)、相对布局(Relative Layout)、表格布局(Table Layout)、绝对布局(Absolute Layout)、框架布局(Frame Layout)等。

Android 中的布局管理一般在 XML 中进行规划和设计,这种方法的优点是直观、简洁,实现了 UI 界面和 Java 逻辑代码的分离。

1. 定义布局文件

在资源文件夹 res\layout 中定义一个 XML 布局文件(如 activity_main. xml),在其中声明布局方式。以下示例代码是一个在 activity_main. xml 中的声明布局的实例,该例采用线性布局,布局中包括一个 TextView 控件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tvHello"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
```

```
        />
</LinearLayout>
```

上述示例代码中出现了一些在 XML 文件中经常使用的布局属性，以下给出简要的说明。

1）ID 属性

控件的 ID 属性可用于在 Java 代码中引用这个相应的控件，当 Java 程序被编译时，这个 ID 作为一个整数被引用，但通常是在 XML 布局文件中以字符串的形式定义一个 ID。一般在 XML 中采用如下方式为其分配 ID 号：android:id="@+id/ID 字符串"，其中在"@+id/"后面的 ID 字符串是设定的 ID 号。这里的@表示 XML 解析器应该解析 ID 字符串并把它作为 ID 资源，当引用此字符串时，解析器会从工程的资源文件夹中读取此值并进行替换；+表示这是一个新的资源名字，它被创建后应加入到资源文件 R. java 中。在 Java 中可以通过调用 Activity. findViewById()方法引用相应的 ID，并创建这个 View 对象的实例。

在定义资源之前一定要先使用 android:id 属性定义它的 ID 号，这样这个资源才能被记录到 R. java 中，之后才能在 Java 源码中使用。

2）尺寸参数

尺寸参数一般是指诸如 layout_height、layout_width 等参数。在表示尺寸时，可用确定的数字（如 50px），也可以采用参数 fill_parent 或者 wrap_content，fill_parent 强制性地使控件扩展，以填充布局单元内尽可能多的空间，如设置一个顶部布局或控件为 fill_parent 将强制性让它布满整个屏幕。wrap_content 则会使视图恰好显示全部内容。以 TextView 控件为例，设置为 wrap_content 将恰好完整显示其内部的文本，布局元素将根据内容自动更改大小。

3）xmlns:android 属性

该属性一般取值为 http://schemas. android. com/apk/res/android。这是定义了一个 XML 命名空间，告诉 Android 开发工具准备使用 Android 命名空间里的一些通用属性。在所有 Android XML 设计文件中最外层的标记必须指定这个命名空间属性。用于标示命名空间的地址不会被解析器用于查找信息。其唯一的作用是赋予命名空间一个唯一的名称。

2. 在 Activity 中引用布局

定义了 XML 布局文件之后，在 Java 程序中调用 setContentView()方法引用这个布局文件，就可以将布局中的控件实例化，如 setContentView(R. layout. activity_main. xml)。

以下示例代码演示了如何在一个 Activity 中使用 activity_main. xml 中的布局，并按布局中设定的内容显示出来。

```
import android.app.Activity;
import android.os.Bundle;
public class TextViewSimple_MainActivity extends Activity {
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
                        //采用 activity_main.xml 文件中的布局
        ((TextView)findViewById(R.id.tvHello)).setText("hello, Activity!");
                        //将字符串"hello, Activity"显示在 ID 号为 tvHello 的 TextView 控件中
    }
}
```

上例的运行结果如图 4-2 所示。

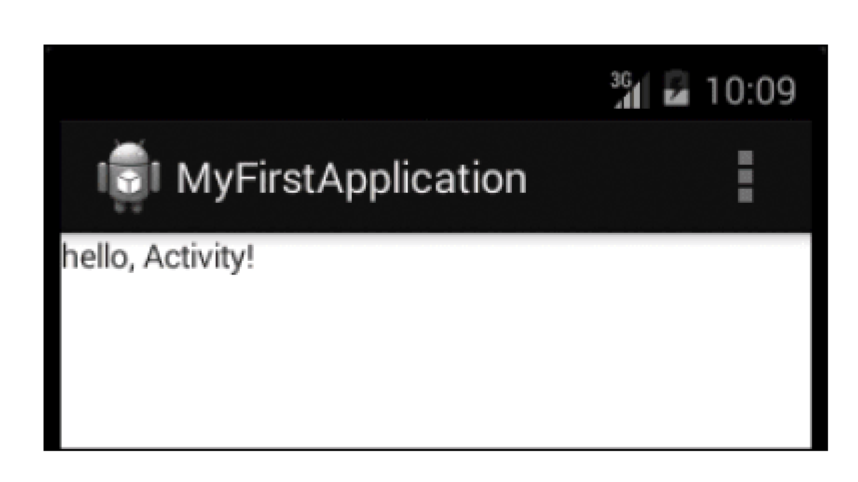


图 4-2　运行结果

4.1.3 线性布局

线性布局的子控件定义在<LinearLayout></LinearLayout>标签之间。线性布局是最简单的布局之一，它将其包含的 Widget 控件元素按水平或者垂直方向顺序排列。布局方向由属性 orientation 的值来决定，android:orientation="vertical"时，子控件垂直排列；android:orientation="horizontal"时，子控件水平排列。同时，使用此布局时可以通过设置控件的 weight 参数控制各个控件在容器中的相对大小。

在线性布局中可使用 gravity 属性来设置控件的对齐方式，例如，对齐到容器左侧(left)、对齐到容器中央位置(center)等。gravity 属性的可供取值有 top(到容器顶)、bottom(底)、left(左)、right(右)、center_vertical(纵向中央)、center_horizonal(横向中央)、center(中央)、fill_vertical(纵向拉伸以填满容器)、fill_horizonal(横向拉伸以填满容器)、fill(纵向横向同时拉伸以填满容器)等。当需要为 gravity 设置多个属性值时，要用“|”将每个属性值分隔。

上述布局的属性既可以在 XML 布局文件中设置，也可以在 Java 程序中通过调用 setOrientation(int)方法和 setGravity(int)方法进行设置。

需要特别注意的是，每一个线性布局的所有子控件，如果垂直分布则仅占一列，如果水平分布则仅占一行。线性布局时如果子控件所需位置超过一行或一列，不会自动换行或换列，超出屏幕的子控件将不会被显示，除非将其放到 ScrollView 中。

【例 4-1】 工程 04_LinearLayoutExample 演示了线性布局的用法。

在 Eclipse 中新建一个工程项目 04_LinearLayoutExample，在项目文件夹下的 res/layout 目录下新建布局文件 linear_layout. xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="right"  >
    <TextView
        android:id="@+id/txtMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="线性布局示例"  />
    <Button
        android:id="@+id/btnYes"
        android:text="是"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"  />
    <Button
        android:id="@+id/btnNo"
        android:text="否"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"  />
    <Button
        android:id="@+id/btnCancel"
        android:text="取消"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"  />
</LinearLayout>
```

linear_layout.xml 布局文件中声明了一个 LinearLayout 布局，并设置其属性为垂直排列、在其所属的父容器中的布局方式为横向和纵向填充父容器、内部元素的对齐方式为向右对齐。

在<LinearLayout>标签中声明了 4 个子控件，分别是一个 TextView 控件和 3 个 Button 控件。TextView 控件占据的空间为恰好包住文字，显示文本内容是“线性布局示例”。3 个 Button 控件占据的空间也是恰好包住文字，按钮上显示文本分别是“是”、“否”和“取消”。

示例程序中使用这个布局文件的是 MainActivity 类，其代码如下。本例中仅仅在 onCreate()方法中调用了 setContentView(R.layout.linear_layout)方法，将 XML 文件中的布局实例化。

```
package edu.hebust.zxm.linearlayoutexample;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.linear_layout);
        }
}
```

上例的运行结果如图 4-3 所示。

把上例 linear_layout.xml 布局文件中的 android:orientation="vertical" 改为 android:orientation="horizontal",则所有控件采用水平布局,其运行结果如图 4-4 所示。



图 4-3　线性垂直布局的运行结果



图 4-4　线性水平布局的运行结果

4.1.4　表格布局

表格布局的子控件定义在<TableLayout></TableLayout>标签之间。

表格布局是一种以类似表格的方式显示 ViewGroup 中元素的布局,它将包含的元素以行和列的形式进行排列,但它并没有表格线,而是采用行和列标识位置。表格布局是 TableLayout 类的实例,一个 TableLayout 由许多的“行”组成。行可以是一个 TableRow 对象,也可以是一个 View 对象。当行是一个 View 对象时,该 View 对象将跨越该行的所有列。

一般在<TableLayout></TableLayout>中间定义<TableRow></TableRow>元素,每个 TableRow 代表一个“行”,拥有零个或多个的“表格单元”cell。每个 cell 又拥有 View 对象,在其中可添加 Widget 控件,如<TextView>、<Button>或<ImageButton>等,

在 TableRow 中可以添加子控件,每添加一个子控件为一列。TableLayout 中可以有空的单元格,也可以有跨越多个列的单元格。在 TableLayout 布局中,一个列的宽度由该列中最宽的那个单元格决定,而表格的宽度是由父容器决定的。要特别注意的是,行号和列号是从 0 开始的。

在 TableLayout 中,可以为列设置 3 种属性。

(1) Shrinkable:如果一个列被标识为 shrinkable,则该列的宽度可以进行收缩,以使表格能够适应其父容器的大小。

（2）Stretchable：如果一个列被标识为 stretchable，则该列的宽度可以进行拉伸，以使填满表格中空闲的空间。

（3）Collapsed：如果一个列被标识为 collapsed，则该列将会被隐藏。

TableLayout 继承自 LinearLayout 类，除了继承来自父类的属性和方法，TableLayout 类中还包含表格布局所特有的属性和方法。例如 android:layout_span，用于设置该控件所跨越的列数。

【例 4-2】 工程 04_TableLayoutExample 演示了表格布局的用法。

在 Eclipse 中新建一个工程项目 04_TableLayoutExample，在项目文件夹下的 res/layout 目录下新建布局文件 table_layout. xml。在布局文件 table_layout. xml 中定义 2 个表格，第一个表格中包含 3 行，第二个表格中包括 2 行。table_layout. xml 中的代码如下。Activity 中的代码与上例类似，在 onCreate()方法中调用了 setContentView(R. layout. table_layout)方法，将 XML 文件中的布局实例化。

```
<?xml version="1.0" encoding="utf-8"?>
<!--声明一个垂直排列的线性布局,里面有两个 table -->
<LinearLayout
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical">

<!--声明第一个表格布局,ID:TableLayout01,里面有三行 -->
    <TableLayout
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#dddddd">
         <!--声明第一行,只有一个 TextView-->
        <TextView
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="第一个 Table 的第一行,TextView 控件独占"
            android:textColor="#000000"
            android:layout_marginTop="4dip" />
        <!--声明第二行,TableRow 包括两列-->
        <TableRow
            android:id="@+id/TableRow01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dip">
            <TextView
```

```
                android:text="短字符串"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true"
                android:textColor="#0000ff"     />
            <TextView
                android:text="第 2行第 2列"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true"
                android:textColor="#000000"   />
        </TableRow>
        <!--声明了第三行,TableRow,ID为 TableRow02-->
        <TableRow   android:id="@+id/TableRow02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="4dip"   >
            <TextView
                android:text="长……字符串"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textColor="#ff0000" />
            <TextView
                android:text="第 3行第 2列"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textColor="#000000"      />
            <TextView
                android:text="第 3行第 3列"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textColor="#000000"   />
        </TableRow>
    </TableLayout>

<!--声明第二个表格布局,ID:TableLayout02,里面有 6个按钮 -->
    <TableLayout
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:shrinkColumns="0,1,2"
        android:collapseColumns="2">
            <!--第一行定义为居中显示,设置 3个按钮-->
      <TableRow android:gravity="center">
```

```
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 1-1"/>
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 1-2" />
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 1-3"/>
          </TableRow><!---->
                <!--第二行定义为居左显示-->
        <TableRow android:gravity="left">
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 2-1"   />
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 2-2"   />
            <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="按钮 2-3"   />
          </TableRow>
      </TableLayout>
</LinearLayout>
```

上例的运行结果如图 4-5 所示。在第 2 个表格中，android:shrinkColumns 属性的作用是设置表格的列是否收缩，多列用逗号隔开。本例中的设置为 android:shrinkColumns＝"0,1,2"，即表格的第 1、2、3 列内容是收缩的，以适合屏幕大小即不会挤出屏幕。android:collapseColumns 属性的作用是设置表格的列是否隐藏，该属性设置为 2，所以第 3 列的按钮没有显示出来。

图 4-5　表格布局的运行结果

4.1.5　相对布局

相对布局的子控件定义在<RelativeLayout></RelativeLayout>标签之间。

在相对布局中，子控件的位置是相对兄弟控件或父容器而决定的，例如，在给定视图的左边或者下面，或相对于某个特定区域的位置（如底部对齐、中间偏左）等来定位元素。在设计相对布局时要按照控件之间的依赖关系排列，如 View A 的位置相对于 View B 来决定，则需要保证在布局文件中 View B 在 View A 的前面。还需要注意的是在进行相对布局时要避免出现循环依赖，例如设置相对布局在父容器中的排列方式为 WRAP_

CONTENT，就不能再将相对布局的子控件设置为 ALIGN_PARENT_ BOTTOM。因为这样会造成子控件和父控件相互依赖和参照的错误。

线性布局和相对布局的区别：线性布局中的控件元素的水平或垂直方向都是线性对齐的，可使用 android:gravity 属性调整向左、右或居中对齐，或使用 android:weight 属性调整其高度或宽度，也可以使用 android:padding 属性来微调各对象的摆放位置；相对布局可以单独指定某个 Layout 或某个对象对齐到另一个 Layout 或对象的位置，而不必像线性布局一样必须将所有的 Layout 与对象对齐。

在进行相对布局时用到的属性很多，很多属性都与位置和距离方式有关，限于篇幅，具体请参阅相关 API 文档。

【例 4-3】 工程 04_RelativeLayoutExample 演示了相对布局的用法。

在 Eclipse 中新建一个工程项目 04_RelativeLayoutExample，在项目文件夹下的 res/layout 目录下新建布局文件 relative_layout.xml，代码如下。在 Activity 的 onCreate()方法中调用 setContentView(R.layout.relative_layout)方法，将 XML 文件中的布局实例化。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
   >
<!--声明了一个 TextView 控件,ID 为 txtMessage-->
    <TextView
        android:id="@+id/txtMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="相对布局示例"
        />
<!--ID 为 btnYes 的按钮,位于 ID 为 txtMessage 的控件下面-->
<!--该控件位于父控件的中央位置。-->
    <Button
        android:id="@+id/btnYes"
        android:text="是"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/txtMessage"
        android:layout_centerInParent="true"
        />
<!--ID 为 btnNo 的按钮,位于 ID 为 btnYes 的组件下面,给定距左的边界-->
    <Button
        android:id="@+id/btnNo"
        android:text="否"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/btnYes"
            android:layout_marginLeft="12dip"
            />
<!--ID为 btnCancel 的按钮,此按钮位于 btnYes 的右边,和 btnYes 的底边齐平-->
        <Button
            android:id="@+id/btnCancel"
            android:text="取消"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/btnYes"
            android:layout_alignBottom="@id/btnYes"
              />
</RelativeLayout>
```

相对布局中，android:layout_centerInParent 属性指定是否位于父控件的中央位置，android:layout_below 属性指定位置为在参照控件的下方，android:layout_toRightOf 属性指定位置为在参照控件右侧，android:layout_alignBottom 属性指定与参照控件底部对齐。

在本例中，“是”按钮位于 TextView 的下方，居中，“否”按钮位于“是”按钮下方，且左边距为 12dip，“取消”按钮位于“是”按钮的右方，运行结果如图 4-6 所示。



图 4-6　相对布局的运行结果

4.1.6　绝对布局

绝对布局的子控件定义在<AbsoluteLayout></AbsoluteLayout>标签之间。

绝对布局是指屏幕中所有控件的位置由开发人员通过设置控件的坐标来指定，即以坐标的方式来定位控件在屏幕上的位置。控件容器不再负责管理其子控件的位置。但通过坐标确定元素位置后，系统无法根据不同屏幕大小对元素位置进行调整，降低了布局对不同类型和尺寸屏幕的适应能力。

【例 4-4】　工程 04_AbsoluteLayoutExample 演示了绝对布局的用法。

在 Eclipse 中新建一个工程项目 04_AbsoluteLayoutExample，在项目文件夹下的 res/layout 目录下新建布局文件 absolute_layout.xml。在 Activity 的 onCreate()方法中调用了 setContentView(R.layout.absolute_layout)方法，将 XML 文件中的布局实例化。

在本例中，布局文件首先定义了<AbsoluteLayout>布局，在其中定义了 2 个用来显示文字的 TextView 控件，一个用于输入的 EditText 控件，一个 OK 按钮。并分别采用绝对坐标方式指定了控件的位置。absolute_layout.xml 文件的代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"  >
    <TextView
        android:layout_x="20dip"
        android:layout_y="20dip"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/TextView01"
        android:text="绝对布局示例"  />
    <TextView
        android:layout_x="40dip"
        android:layout_y="40dip"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:id="@+id/TextView02"
        android:text="控件:TextView02"  />
    <EditText
        android:layout_x="60dip"
        android:layout_y="60dip"
        android:layout_height="90dip"
        android:layout_width="180dip"
        android:id="@+id/EditText01"
        android:text="控件:EditText01"  />
    <Button
        android:layout_x="200dip"
        android:layout_y="160dip"
        android:layout_height="wrap_content"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:text="OK"  />
</AbsoluteLayout>
```

上例的运行结果如图 4-7 所示。



图 4-7 绝对布局的运行结果

4.1.7 框架布局

框架布局的子控件定义在<FrameLayout></FrameLayout>标签之间。

FrameLayout 布局在屏幕上开辟出一块区域，在这块区域中可以添加多个子控件，但是所有的子控件都被对齐到区域的左上角。框架布局的大小由子控件中尺寸最大的那个子控件

来决定。如果子控件一样大，同一时刻只能看到最上面的子控件。

框架布局使多个组件以层叠的效果呈现给用户。应用程序的布局采用框架布局时，控件元素的位置只能放置在显示空间的左上角而无法指定到一个确切的位置。如果有多个元素，后放置的元素将遮挡先放置的元素。

【例 4-5】 工程 04_FrameLayoutExample 演示了框架布局的用法。

在 Eclipse 中新建一个工程项目 04_FrameLayoutExample，在项目文件夹下的 res/layout 目录下新建布局文件 frame_layout.xml。在 Activity 的 onCreate()方法中调用 setContentView(R.layout. frame_layout)方法，将 XML 文件中的布局实例化。

在本例中，布局文件首先定义了<FrameLayout>，在其中定义了 3 个用来显示文字的 TextView 控件。frame_layout.xml 文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<TextView
        android:text="较大的文字"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="26pt"
        android:textColor="#dddddd"/>
<TextView
        android:text="中等的文字"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18pt"
        android:textColor="#aaaaaa"/>
<TextView
        android:text="较小的文字"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:textColor="#000000"/>
</FrameLayout>
```

上例的运行结果如图 4-8 所示。程序运行时所有的子控件都自动地对齐到容器的左上角，由于子控件的 3 个 TextView 是按照字号从大到小排列的，所以字号小的在最上层。



图 4-8　框架布局的运行结果

4.2 Widget 控件

Android 中的 Widget 是 Android SDK 1.5 之后新加入的一个开发框架。作为一组用于绘制交互屏幕元素的类，相当于 Windows 应用程序中的小插件，可以嵌入到手机应用程序中的人机交互界面上。Widget 控件都在 android.widget 包中，用户在编程时可以直接使用。在进行用户界面设计时这些控件是必不可少的重要元素。

本节介绍文字控件 TextView、文字输入框 EditText、按钮 Button、复选框 CheckBox 和单选按钮 RadioButton。第 5 章将介绍其他常用的 Widget 控件。很多 Widget 控件既可以在 XML 文件中设定各种属性，也可以在 Java 文件中设定属性。如果需要动态地改变某些属性值，例如 TextView 中的文本，通常要在 Java 代码中实现。

4.2.1 TextView 和 EditText

TextView 通常用于在 Activity 上设置显示文字，而 EditText 通常用于在 Activity 上接受用户从键盘输入的文本信息。TextView 常用于 EditText 的前面，作为文本输入框的提示信息。

EditText 是 TextView 的子类，所以 TextView 的方法和属性同样适用于 EditText。EditText 控件还具有一些与 TextView 不同的属性，如以密码方式显示(其 password 属性为 true，即 android:password="true")、设定其 hint 提示信息等。调用 selectAll()方法可以选中 EditText 中的全部文字。

TextView 和 EditText 的创建与使用方法类似，通常的使用步骤如下。

步骤 1：在 XML 布局文件中定义 TextView 或 EditText，可以设置其 ID 属性、显示文字的内容、宽度和高度等。此外还可以设置字体、内容、颜色等属性。

步骤 2：在 Activity 中声明 TextView 或 EditText 实例对象。

步骤 3：利用 findViewById()方法获取 XML 布局文件中定义的 TextView 或 EditText。在 Java 程序中可以调用 setText()方法设置文字内容到控件中，或调用 getText()方法取得控件中的文字。

【例 4-6】 工程 04_TextExample 演示了 TextView 和 EditText 控件的使用方法。

在 Eclipse 中新建一个工程项目 04_TextExample，在资源文件夹 res\layout 中定义一个 XML 布局文件，本例中文件名为 activity_main.xml，布局中包括 3 个 TextView、一个 EditText。第一个 TextView 用于演示在 XML 布局文件中定义文字的属性；第二个 TextView 用于文本框的输入提示；第三个 TextView 用于演示在 Java 文件中定义文字的属性。activity_main.xml 文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"   >
```

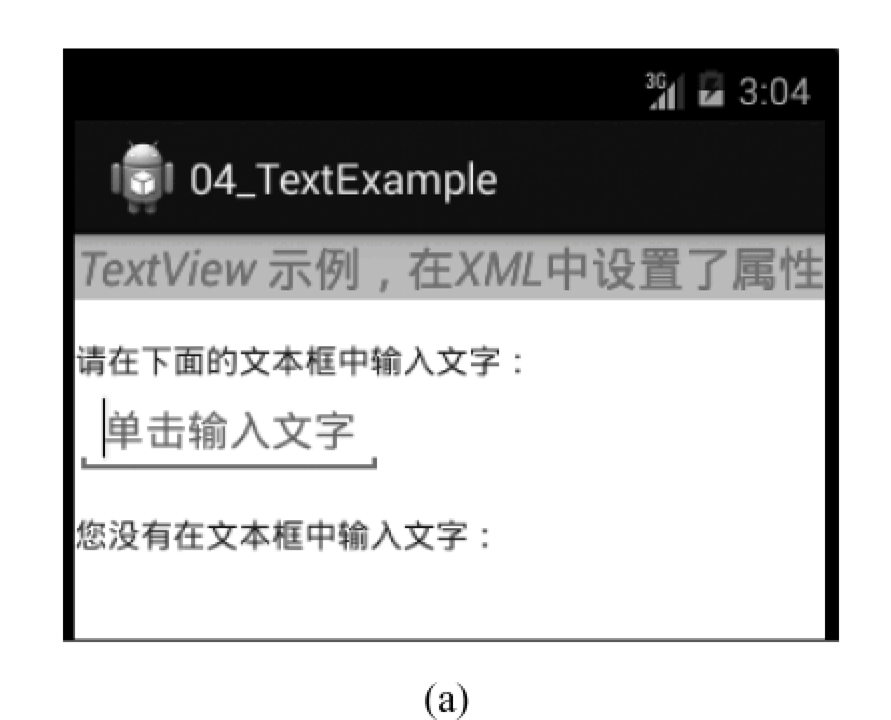
```
    <TextView
        android:id="@+id/tv_hello"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="TextView 示例,在 XML 中设置了属性"
        android:textColor="#ff00ff"
        android:textStyle="italic"
        android:background="#cccccc"
        android:textSize="20sp"
        android:gravity="center_horizontal"   />
    <TextView
        android:id="@+id/tv_message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="\n 请在下面的文本框中输入文字:" />
    <EditText
        android:id="@+id/txt_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:text=""
        android:hint="单击输入文字"/>
    <TextView
        android:id="@+id/tv_txtout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="\n 您没有在文本框中输入文字:" />
</LinearLayout>
```

下面的示例代码演示了如何在一个 Activity 中使用 activity_main. xml 中的 TextView 和 EditText 控件。代码中处理了 EditText 的按键事件,当用户在文本框中输入字符时,更新其下面 TextView(ID: tv_txtout)控件显示的文字内容。EditText 按键事件的处理机制和方法详见本章 4.3 节。

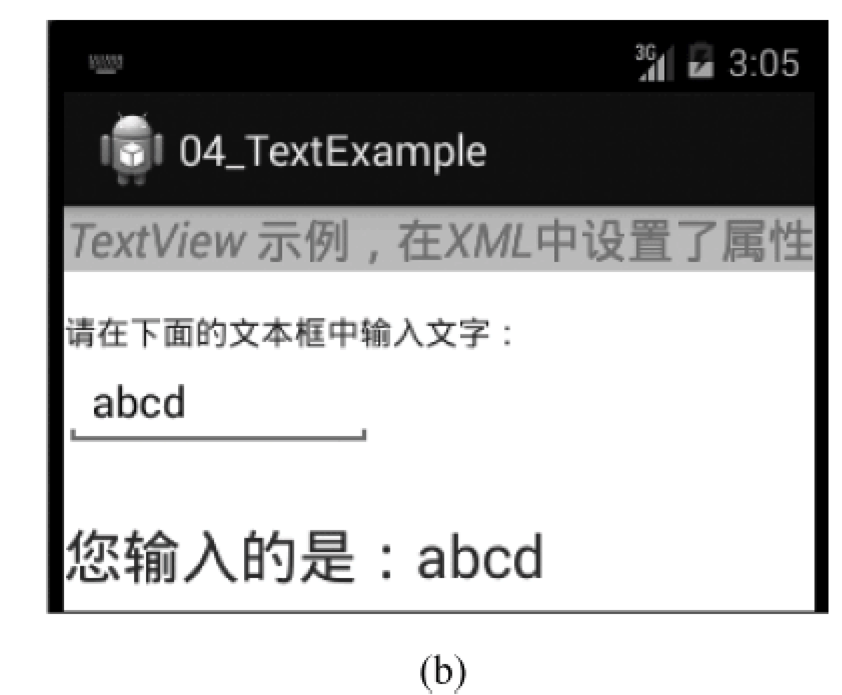
```
//package 和 import 语句略
public class MainActivity extends Activity {
    TextView TxtOut;
    EditText TxtIn;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /* 以下在 Java 代码中获取 XML 布局文件中定义的控件 */
        TxtOut= (TextView)findViewById(R.id.tv_txtout);
        TxtIn= (EditText)findViewById(R.id.txt_input);
```

```
            TxtIn.setOnKeyListener(new OnKeyListener() {
                public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
                    /* 取得 EditText 控件中的文字 */
                    String str=TxtIn.getText().toString();
                    /* 以下在 Java 代码中设置 TextView 的属性 */
                    TxtOut.setTextColor(Color.BLUE);
                    TxtOut.setTextSize(25);
                    TxtOut.setText("\n您输入的是:"+str);
                    return false;
                }
            });
        }
    }
```

上例的运行结果如图 4-9 所示。



(a)



(b)

图 4-9　04_TextExample 的运行结果

注意上述代码中第三个 TextView 在 XML 布局文件中预设的显示文字为"您没有在文本框中输入文字："，但当用户输入文字后，却显示"您输入的是：×××"，文字的大小和颜色也发生了变化，这是因为通过调用 TextView 类的 setText()方法重新设定了 TextView 的字符内容，调用 setTextColor()和 setTextSize()方法重新设定了文字的颜色和大小。

4.2.2　Button

Button 继承自 android.widget.TextView，是 UI 中的基本元素。在 XML 布局文件中可以添加及设定 Button 的位置、形态、显示文字等。Button 的创建与使用方法与前述的 TextView 类似，通常的使用步骤如下。

步骤 1：在 XML 布局文件中定义 Button，设置其 ID 属性、按钮上显示文字的内容、宽度和高度等属性。

步骤 2：在 Activity 中声明 Button 实例对象。

步骤 3：利用 findViewById()方法获取 XML 布局文件中定义的 Button。对于按钮

控件，通常需要设计其单击后的处理逻辑，一般要在 Activity 类中通过监听相应的事件来进行处理。Android 中的事件处理机制详见 4.3 节。

【例 4-7】 工程 04_ButtonExample 演示了 Button 控件的使用方法。

在 Eclipse 中新建一个工程项目 04_ButtonExample，在项目文件夹下的 res/layout 目录下新建布局文件 activity_main.xml。布局中包括一个 TextView 和两个 Button，第二个 Button 添加了系统样式。设置了一个 style 样式属性，此属性通过调用系统的内置样式文件设置 Button 的大小和文字样式等。格式为“? [package:][type:]name”。

activity_main.xml 文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv_button"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="显示两个按钮"
        />
    <Button android:id="@+id/btn_normal"
        android:text="普通按钮"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/btn_small"
        style="?android:attr/buttonStyleSmall"
        android:text="小小按钮"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

以下示例代码演示了如何在一个 Activity 中使用 activity_main.xml 中的 Button 控件。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button normal= (Button)findViewById(R.id.btn_normal);
            //找到 ID 号为 btn_normal 的按钮
        Button small= (Button)findViewById(R.id.btn_small);
            //找到 ID 号为 btn_small 的按钮
```

```
    }
}
```

上例的运行结果如图 4-10 所示。



图 4-10　04_ButtonExample 的运行结果

4.2.3　CheckBox

CheckBox 继承自 android.widget.CompoundButton 类，是一个可以同时选择多个选项的 UI 控件。CheckBox 的使用方法与前面介绍的控件类似，先在 XML 布局文件(如 res\layout\activity_main.xml)中提前定义并设置属性，然后通过在 Activity 类中调用 setContentView()方法来使用它。

【例 4-8】 示例工程 04_CheckBoxExample，演示 CheckBox 控件的用法。

在 Eclipse 中新建一个工程项目 04_CheckBoxExample，在项目文件夹下的 res/layout 目录下新建布局文件 activity_main.xml。布局中包括一个 TextView 和 4 个 CheckBox，第一个选项设置为选中状态。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/favouriteLabel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请勾选您感兴趣的图书类别" />
    <CheckBox
        android:id="@+id/cbox_classical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:checked="true"
        android:text="古典文学" />
        <!--设置 android:checked 属性值为"true",将该选项设置为选中状态-->
```




```
    <CheckBox
        android:id="@+id/cbox_novel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="当代小说" />
    <CheckBox
        android:id="@+id/cbox_essays"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="散文随笔" />
    <CheckBox
        android:id="@+id/cbox_poetry "
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="诗歌" />
</LinearLayout>
```

在下面的示例代码中，MainActivity 使用 activity_main.xml 指定的布局，并按布局中设定的内容显示出来。复选框可以进行默认勾选的设置，如本例将“古典文学”项设定为勾选状态。这可以在 XML 文件中通过设置属性值来实现，也可以使用 Java 程序代码来实现，设置复选框的 Check 状态时，调用 setChecked()方法，参数为 true 时为选中，为 false 时为未选中。在下面的示例代码中将“散文随笔”项设定为勾选状态。

```
//package 和 import 语句略
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        CheckBox  my_essays= (CheckBox)findViewById(R.id.cbox_essays);
                //使 my_essays 指向 id 号为 cbox_essays 的 CheckBox
        my_essays.setChecked(true);                    //设置为选中状态
    }
}
```

上例的运行结果如图 4-11 所示。



图 4-11　04_CheckBox Example 的运行结果

4.2.4　RadioGroup 和 RadioButton

RadioButton 为单选按钮，主要用于多值选一的操作，例如性别的选择，仅能从“男”或“女”中选择一项，那么就可以使用单选按钮实现。在 Android 中实现单选需要用到 Radio-

Group 和 RadioButton 两个视图控件，它们结合使用才能达到单选按钮的效果。

与 CheckBox 相同，RadioButton 也继承自 android.widget.CompoundButton 类。使用 RadioButton 时一般要使用 RadioGroup 来对几个 RadioButton 分组。RadioGroup 是 RadioButton 的承载体，但 RadioGroup 在程序运行时不可见。一个 Activity 中可包含一个或多个 RadioGroup，而每个 RadioGroup 可包含一个或多个 RadioButton。

【例 4-9】 示例工程 04_RadioExample 演示了 RadioGroup 和 RadioButton 的用法。

在 Eclipse 中新建一个工程项目 04_RadioExample，在项目文件夹下的 res/layout 目录下新建布局文件 activity_main.xml。布局中包括一个 TextView 和 4 个 RadioBox，按钮的排列方法是垂直方式，设置第一个 RadioBox 控件为选中状态。activity_main.xml 文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/favourite_single"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请选择一个您感兴趣的图书类别" />
    <RadioGroup
        android:id="@+id/gender"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <RadioButton
            android:id="@+id/rbt_classical"
            android:checked="true"
            android:text="古典文学" />
        <!—设置 android:checked 属性值为"true",将该选项设置为选中状态-->
        <RadioButton
            android:id="@+id/rbt_novel"
            android:text="当代小说" />
        <RadioButton
            android:id="@+id/rbt_essays"
            android:text="散文随笔" />
        <RadioButton
            android:id="@+id/rbt_poetry "
            android:text="诗歌" />
    </RadioGroup>
</LinearLayout>
```

在 Activity 中使用 setContentView(activity_main.xml)方法引用 activity_main.xml 文件的布局，就可以将布局中设定的单选按钮显示出来。与 CheckBox 类似，默认选中按钮的设置，可以在 XML 文件中设置，也可以使用 Java 程序代码调用 setChecked()方法来实现。

上例的运行结果如图 4-12 所示。



图 4-12　04_RadioExample 的运行结果

4.3　Android 中的事件处理机制

在图形用户界面的开发设计中，有两个非常重要的内容：一个是控件的布局，另一个是控件的事件处理。Android 在事件处理过程中主要涉及 3 个概念。

(1) 事件：表示用户在图形界面的操作的描述，通常是封装成各种类，例如，键盘事件操作相关的类为 KeyEvent，触摸屏相关的移动事件类为 MotionEvent 等。

(2) 事件源：事件源是指发生事件的控件，例如，Button、EditText 等。

(3) 事件处理者：事件处理者是指接收事件并对其进行处理的对象，事件处理者一般是一个实现某些特定接口类的对象。

Android 系统的用户与应用程序之间的交互是通过事件处理来完成的，各控件在不同情况下触发的事件可能并不相同，但对事件的处理方法主要有两类，即“基于监听接口”的处理方法和“基于回调机制”的处理方法。前者使用事件监听器 Event Listeners 来处理事件，后者使用 Event Handlers 来处理事件。另外，Android 还提供一种更简单的绑定事件监听器的方式，即直接在界面布局文件中为指定控件绑定事件处理方法。

4.3.1　基于监听接口的事件处理

与 Java 中的监听处理模型一样，Android 也提供了同样的基于监听接口的事件处理模型。基于监听接口的事件处理方法有 3 种实现方式：直接实现接口的处理方式、内部类处理方式和匿名内部类处理方式。下面以响应 ID 为 btn_normal 的按钮单击事件为例介绍这 3 种事件处理方式，事件处理的结果是使 TextView 控件(示例程序中该控件的 ID 为 tv_button)的文本内容变为“你好，normal 按钮被单击！”。

将事件源与事件监听器联系在一起，就需要为事件源注册监听事件，即为事件源对象

添加某个事件的监听。当事件发生时，系统会将事件封装成相应类型的事件对象，并发送给注册到事件源的事件监听器。当监听器对象接收到事件对象之后，会调用监听器中相应的事件处理方法来处理事件，并给出响应。

1. 直接实现接口的处理方式

这种方式定义 Activity 时直接实现接口，这样 Activity 本身就是事件监听器，可以实现事件的监听和响应。

处理按钮点击事件时，一般需要调用该按钮实例的 setOnClickListener()方法，并把 View. OnClickListener 对象的实例作为参数传入。通过侦听按钮被单击的事件，可以完成相应的功能。一般是在 View. OnClickListener 的 onClick()方法里处理按钮的单击事件。

这种方式使用 Activity 本身作为监听器类，可以直接在 Activity 类中定义事件处理器方法，形式非常简洁。但这种做法可能造成程序结构混乱，因为 Activity 的主要职责是完成界面初始化，但此时还需包含事件处理器方法。

【例 4-10】 示例工程 04_ButtonClickExample1 演示了采用直接实现接口的处理方式处理按钮单击事件。

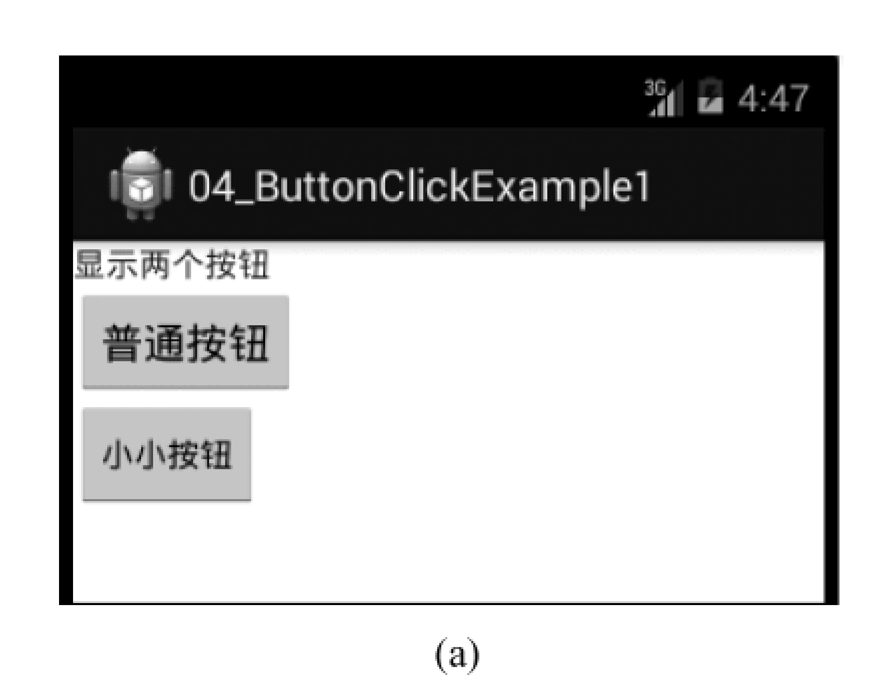
本例使用 4.2.2 节的布局文件。在 Activity 中实现接口，并且绑定在 Button 按钮(ID：btn_normal)上。本例中要监听按钮的单击事件，所以需要实现 OnClickListener 接口，在使用之前要用 import 语句引入 android. view. View. OnClickListener。Activity 代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity implements OnClickListener {
            //定义 MainActivity 类时实现 OnClickListener 接口
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mybutton= (Button)findViewById(R.id.btn_normal);
                                          //定义 Button 按钮
        mybutton.setOnClickListener(this);
        //为事件源对象添加 Click 事件的监听,将事件处理绑定到事件源
    }
    @Override
    public void onClick(View v) {
    //监听器定义的事件处理方法,重写该方法,实现监听事件的处理,参数 v 是事件发生的事
      件源
        switch(v.getId()) {
            case R.id.btn_normal:
                TextView mytext= (TextView)findViewById(R.id.tv_button);
                mytext.setTextColor(Color.BLUE);
```

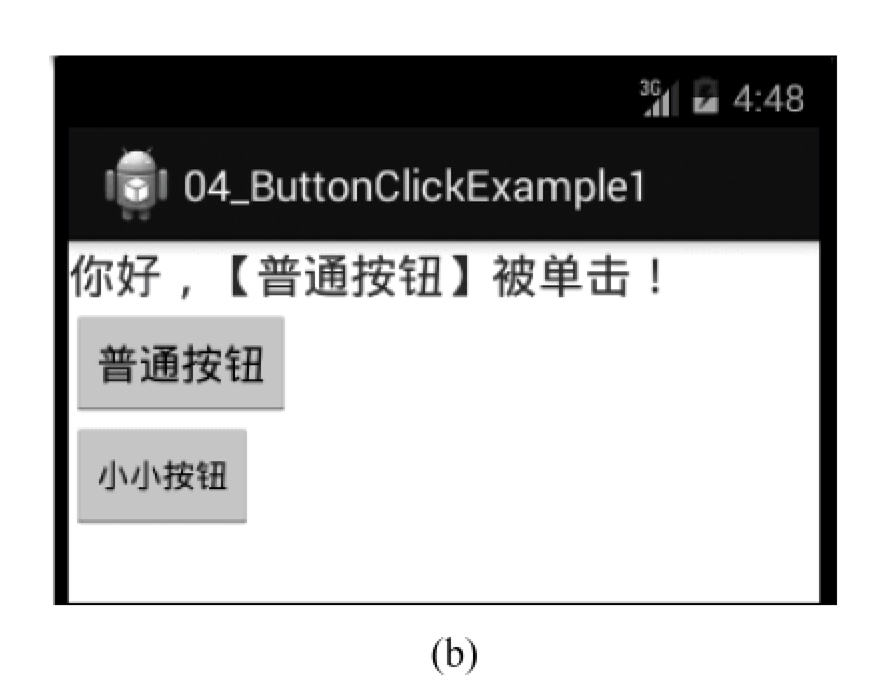



```
                mytext.setTextSize(20);
                mytext.setText("你好,【普通按钮】被单击!");
                break;
            }
        }
    }
```

单击按钮前的程序界面如图 4-13(a)所示,单击按钮后的程序界面如图 4-13(b)所示。



(a)



(b)

图 4-13　工程 04_ButtonClickExample1 的运行结果

从上例可以看出,对事件的处理主要是继承并完成 OnClickListener 接口中的 onClick 方法,并且将其绑定在事件源中,从而达到事件处理的效果。

2. 内部类处理方式

这种处理方式将事件监听器类定义成当前类的内部类。使用内部类可以在当前类中复用监听器类,并且可以自由访问外部类的所有界面组件,这也是内部类的优势。

【例 4-11】 示例工程 04_ButtonClickExample2 演示了采用内部类处理方式处理按钮单击事件。

与工程 04_ButtonClickExample1 类似,使用 4.2.2 节的布局文件。在 Activity 中使用内部类事件处理模型,处理在按钮 btn_normal 的单击事件。本例的处理方式虽然与前例不同,但同样需要实现 OnClickListener 接口,在使用之前要用 import 语句引用 android.view.View.OnClickListener。本例的运行结果与前例相同,Activity 代码如下:

```
//package 和 import 语句略
public class MainActivity extends Activity {
    //不需要在定义 MainActivity类时实现 OnClickListener接口
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mybutton= (Button)findViewById(R.id.btn_normal);
        mybutton.setOnClickListener(new MyClickListener());
```

```
            //为事件源对象添加 Click 事件的监听,将事件处理绑定到事件源
            //参数是一个内部类 MyClickListener 的对象,这个类实现 OnClickListener 接口
        }
        //定义内部类 MyClickListener,实现 OnClickListener 接口
        class MyClickListener implements OnClickListener {
            @Override
        public void onClick(View v) {                //同样,重写该方法实现监听事件的处理
                switch(v.getId()) {
                    case R.id.btn_normal:
                        TextView mytext= (TextView)findViewById(R.id.tv_button);
                        mytext.setText("你好,【普通按钮】被单击!");
                        break;
                }
            }
        }
    }
```

说明：将 MyClickListener 类设计成一个普通的外部类,也能实现此功能。但这种定义事件监听器类的形式比较少见,主要有两个原因：首先,事件监听器通常属于特定的 UI 界面,定义成外部类不利于提高程序的内聚性。其次,外部类形式的事件监听器不能自由访问创建 UI 界面的类中的组件,编程不够简洁。但如果某个事件监听器确实需要被多个 UI 界面所共享,而且主要是完成某种业务逻辑的实现,则可以考虑使用外部类的形式来定义事件监听器类。

3. 匿名内部类处理方式

使用匿名内部类处理方式是一种最常用的方法,因为大部分事件监听器只是临时使用一次,所以使用匿名内部类形式的事件监听器更合适。

【例 4-12】 示例工程 04_ButtonClickExample3 演示了采用匿名内部类处理方式处理按钮单击事件。

将例 4-11 中的事件处理过程通过匿名内部类实现,在 Activity 中的事件处理代码有所不同,其代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button mybutton= (Button)findViewById(R.id.btn_normal);
        final TextView mytext= (TextView)findViewById(R.id.tv_button);
        mybutton.setOnClickListener(new OnClickListener() {                  //匿名内部类
            public void onClick(View v) {      //参数 v 就是事件发生的事件源
                switch(v.getId()) {
                    case R.id.btn_normal:
```




```
                        mytext.setText("你好,【普通按钮】被单击!");
                        break;
                    }
                }
            });
        }
}
```

以上 3 种方式都是实现接口中的 onClick 方法，并且绑定于特定的事件源，从而达到事件的处理。其中，接口实现方法和内部类都是通过继承接口 OnClickListener 并重写其中的 onClick()方法；匿名内部类实现方法则是通过重写 onClick()方法。

4.3.2 基于回调机制的事件处理

在 Android 中任何一个控件和 Activity 都是间接或者直接继承于 android. view. View，几乎每个 View 都有自己的处理事件的回调方法，开发人员可以通过重写 View 中的这些回调方法来实现对事件的响应。当某个事件没有被任何一个 View 处理时，便会调用 Activity 中相应的回调方法。

1. onKeyDown()方法

onKeyDown()方法是接口 KeyEvent. Callback 中的抽象方法，用于捕获按键信息并对其处理。所有的 View 全部实现了该接口并重写了该方法，该方法用于捕获设备键盘被按下的事件，其定义如下：

```
public boolean onKeyDown(int KeyCode, KeyEvent event)
```

其中，参数 KeyCode 为被按下的键盘码，设备键盘中每个按钮都会有其单独的键盘码，应用程序都是通过键盘码知道用户按下的是哪个键。参数 event 是按键事件对应的对象，其中包含了触发事件的详细信息，例如事件的状态、事件的类型、事件发生的时间等。当用户按下按键时，系统会自动将事件封装成 KeyEvent 对象供应用程序使用。

该方法的返回值是一个 boolean 类型的值，返回 true 时表示已经完整地处理了事件并不希望其他回调方法再次处理，而返回 false 时表示并没有完全处理完该事件并希望其他回调方法继续对其进行处理。

【例 4-13】 示例工程 04_OnKeyDownExample 演示了通过 onKeyDown()方法来监听被按下的按键信息并将其显示到 TextView 中。Java 代码中必须要用 import 语句引入 android. view. KeyEvent。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
    }
    @Override                                          //重写 onKeyDown()方法
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        Time t=new Time();
        t.setToNow();                                  //得到当前时间
        TextView mytext=(TextView)findViewById(R.id.tv_message);
        mytext.setTextSize(15);                        //设置文字大小
        mytext.setText("当前时间是："+t.toString()+"\n键盘码："+keyCode);
        return super.onKeyDown(keyCode, event);
    }
}
```

程序运行后，当按下某键时，显示按键时间和键盘码，运行结果如图 4-14 所示。

类似地，onKeyUp()方法可以用来捕捉设备键盘按键抬起的事件，其参数和使用方法与 onKeyDown()类似，在此不再赘述。



图 4-14 工程 04_OnKeyDownExample 的运行结果

2. onTouchEvent()方法

onTouchEvent()是在 View 中定义的一个方法，用于捕获触摸屏事件并对其进行处理。有的 View 子类全部重写了该方法。onTouchEvent()处理传递到 View 的手势事件，包括 ACTION_DOWN(屏幕被按下)、ACTION_MOVE(在屏幕中拖动)、ACTION_UP(屏幕被抬起)、ACTION_CANCEL 4 种事件。Android 系统支持触摸屏操作，应用程序可以通过该方法处理移动设备屏幕的触摸事件。

onTouchEvent()方法的定义如下：

```
public boolean onTouchEvent(MotionEvent event)
```

其中，参数 event 为手机屏幕触摸事件封装类的对象，它封装了该事件的所有信息，如触摸的位置、类型以及触摸的时间等。该对象会在用户触摸手机屏幕时被创建。onTouchEvent()方法的返回值与 onKeyDown()等方法相似，是当已经完整地处理了该事件且不希望其他回调方法再次处理时返回 true，否则返回 false。

一般情况下，当屏幕被按下、触控笔离开屏幕、触控笔在屏幕上滑动 3 种事件全部由 onTouchEvent()方法处理。onTouchEvent()方法捕捉到这些事件后，调用 MotionEvent.getAction()方法来获取动作值，判断发生的是哪一个事件，然后分别对其处理。MotionEvent.getAction()的值为 MotionEvent.ACTION_DOWN，处理屏幕被按下的事件；MotionEvent.getAction()的值为 MotionEvent.ACTION_UP 时，处理屏幕被抬起的事件；MotionEvent.getAction()的值为 MotionEvent.ACTION_MOVE，处理触控笔在屏幕上滑动事件。该方法一般出现在 Activity 中的位置是在 onCreate()方法之后，在重写 public boolean onTouchEvent(MotionEvent event)方法时，根据侦听到的不

同情况分别处理。

【例 4-14】 工程 04_OnTouchEventExample 演示了通过 onTouchEvent()方法来监听触摸屏事件并将触摸点坐标信息显示到 TextView 中。

Android 系统的坐标系与 Java 相同,以左上顶点为原点坐标(0,0),向右为 X 抽正方向,向下为 Y 抽正方向。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    TextView TxtAction,TxtPostion ;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TxtAction= (TextView)findViewById(R.id.tv_action);
        TxtPostion= (TextView)findViewById(R.id.tv_postion);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        int Action=event.getAction();
        float x=event.getX();
        float y=event.getY();
        TxtAction.setText("触屏动作:"+Action);
        TxtPostion.setText("当前坐标:"+"("+x+" , "+y+")");
        return true;
    }
}
```

示例程序运行后,触摸屏幕的响应结果如图 4-15 所示。

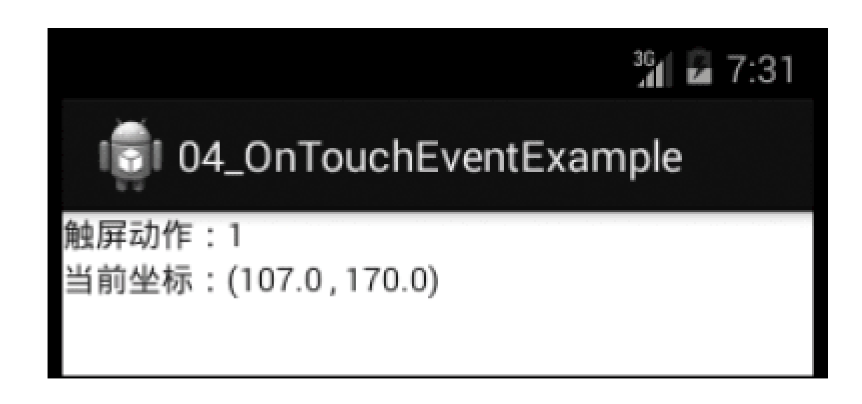


图 4-15　工程 04_OnTouchEventExample 的运行结果

4.3.3　直接绑定到标签的事件处理方法

Android 还提供一种更简单的绑定事件监听器的方式,直接在界面布局文件中为指定控件绑定事件处理方法。对于很多 Android 控件而言,它们都支持如 onClick、onLongClick 等属性,这种属性的属性值就是一个形如 xxx(View source)方法的方法名。例如,以下示例代码为 Button 按钮绑定一个事件处理方法 buttonClick(View source)。

```
<Button android:id="@+id/btn_normal"
    android:text="普通按钮"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="buttonClick"/>
```

编程需要在该界面布局对应的 Activity 中定义一个 void buttonClick(View source)方法,该方法将会负责处理该按钮上的单击事件,代码如下:

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
}

    //定义一个 void buttonClick(View source)方法
    public void buttonClick(View v) {
        switch(v.getId()) {
        case R.id.btn_normal:
            TextView mytext=(TextView)findViewById(R.id.tv_button);
            mytext.setText("你好,【普通按钮】被单击!");
            break;
        }
    }
}
```

示例工程 04_ButtonClickExample4,即采用上述方法处理 Button 按钮(ID:btn_normal)的单击(Click)事件。该工程的源代码见本书的电子配套资源。

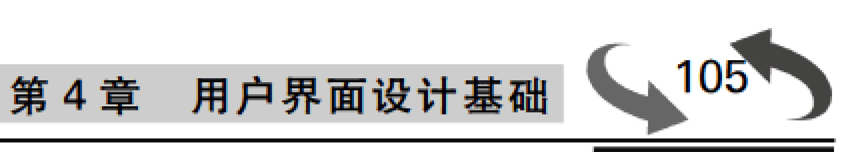
4.3.4 EditText、CheckBox 和 RadioButton 的常见事件处理

EditText 控件通常用于通过键盘输入文字,所以最常用的是监听键盘的按键事件。例如,对用户输入内容时做检查,如果用户输入不合法的内容不予以显示或给出错误提示。使用 setOnKeyListener()可以监听到对 EditText 的按键动作,每按键一次该事件就发生一次。但需要注意的是,这种方式只能监听硬键盘事件。

对于 CheckBox 控件和 RadioButton 控件,其常见事件是选择项发生改变,调用 setOnCheckedChangeListener()方法可以实现该事件的监听和处理。

【例 4-15】 在示例工程 04_EventExample 中,当在 EditText 输入文字、或 RadioButton 被选择或 CheckBox 被选择时,最下方的 TextView 显示相应的结果。

首先在布局文件中定义了一个 EditText、一个 RadioGroup 和 3 个 RadioButton、3 个 CheckBox。之后,在 Activity 中通过 ID 引用相应的控件,监听相关事件并作出相应的处理。



布局文件 activity_main. xml 的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv_message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请在下面的文本框中输入文字:"  />
    <EditText
        android:id="@+id/txt_input"
        android:layout_width="200dip"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:text=""  />
    <TextView
        android:id="@+id/favourite_single"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="请选择您的学历:" />
    <RadioGroup
        android:id="@+id/ rg_education"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <RadioButton
            android:id="@+id/rbt_bachelor"
            android:layout_width="wrap_content
            android:text="大学本科"
            android:checked="true"/>
        <RadioButton
            android:id="@+id/rbt_master"
            android:layout_width="wrap_content"
            android:text="硕士研究生" />
        <RadioButton
            android:id="@+id/rbt_doctor"
            android:layout_width="wrap_content"
            android:text="博士研究生" />
    </RadioGroup>
    <TextView
```

```
        android:id="@+id/favouriteLabel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请勾选您感兴趣的图书类别:" />
    <CheckBox
        android:id="@+id/cbox_classical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="古典文学" />
    <CheckBox
        android:id="@+id/cbox_novel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="当代小说" />
    <CheckBox
        android:id="@+id/cbox_essays"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="散文随笔" />
    <CheckBox
        android:id="@+id/cbox_poetry "
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="诗歌" />
    <TextView
        android:id="@+id/tv_result"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#0000ff"
        android:text="\n您还没有输入任何文字,\n您的学历是:大学本科,\n您感兴趣的图书类
        别:无" />
</LinearLayout>
```

在 Activity 中,使用 setOnKeyListener()方法来捕获针对 EditText 的按键动作。在方法中,通过调用 txtInput.getText()获取用户在 EditText 中的输入,然后将其设置为 TextView 的内容,注意此处类型的转换及其实现方式。对 RadioGroup 调用 setOnCheckedChangeListener()方法进行侦听并处理选择按钮信息,对 CheckBox 调用 setOnCheckedChangeListener()方法侦听并处理复选框被选择的选择信息。

需要注意的是,RadioGroup 使用的监听器是 RadioGroup.OnCheckedChangeListener,而 CheckBox 使用的监听器是 CompoundButton.OnCheckedChangeListener。当在同一个 Activity 中同时存在 RadioGroup 与 CheckBox 且都要监听事件时,这两个监听器就会出现冲突,导致程序不能被编译运行。所以这两个类不能同时使用 import 语句导入。解决方法是在代码中通过完整路径 android.widget.CompoundButton.OnCheckedChangeListener 的方式

使用该监听器。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private String myresults1= "\n 您还没有输入任何文字，",
            myresults2= "\n 您的学历是:大学本科，",
            myresults3= "\n 您感兴趣的图书类别:无";
    TextView tvResult;
    EditText txtInput;
    RadioButton rb1,rb2,rb3;
    RadioGroup rg;
    CheckBox cbox1,cbox2,cbox3,cbox4;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvResult= (TextView)findViewById(R.id.tv_result);
        txtInput= (EditText)findViewById(R.id.txt_input);
        rb1= (RadioButton)findViewById(R.id.rbt_bachelor);
        rb2= (RadioButton)findViewById(R.id.rbt_master);
        rb3= (RadioButton)findViewById(R.id.rbt_doctor);
        rg= (RadioGroup)findViewById(R.id.rg_education);

        //对 EditText 进行监听
        txtInput.setOnKeyListener(new OnKeyListener() {
            @Override
            public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
                String str= txtInput.getText().toString();
                myresults1= "\n 您输入的是:"+ str;
                tvResult.setText(myresults1+ myresults2+ myresults3);
                return false;
            }
        });

        //对 RadioGroup 进行监听
        rg.setOnCheckedChangeListener(new OnCheckedChangeListener() {
          public void onCheckedChanged(RadioGroup group, int checkedId) {
            if(R.id. rbt_bachelor= = checkedId) {
                myresults2= "\n 您的学历是:"+ rb1.getText().toString();
            }
            else if(R.id. rbt_master= = checkedId) {
                myresults2= "\n 您的学历是:"+ rb2.getText().toString();
```

```
                }
                else if(R.id.rbt_doctor==checkedId) {
                    myresults2="\n您的学历是:"+rb3.getText().toString();
                }
                tvResult.setText(myresults1+myresults2+myresults3);
            }
        });
        cbox1=(CheckBox)findViewById(R.id.cbox_classical);
                                        //通过 ID 找到 CheckBox
        cbox2=(CheckBox)findViewById(R.id.cbox_novel);
        cbox3=(CheckBox)findViewById(R.id.cbox_essays);
        cbox4=(CheckBox)findViewById(R.id.cbox_poetry);

        //对 CheckBox 进行监听
        cbox1.setOnCheckedChangeListener(cBoxListener);
        cbox2.setOnCheckedChangeListener(cBoxListener);
        cbox3.setOnCheckedChangeListener(cBoxListener);
        cbox4.setOnCheckedChangeListener(cBoxListener);
    }
    private android.widget.CompoundButton.OnCheckedChangeListener cBoxListener = new android.widget.
    CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
                myresults3="\n您感兴趣的图书类别:";
                if(cbox1.isChecked()) {
                    myresults3=myresults3+"  "+cbox1.getText().toString();
                }
                if(cbox2.isChecked()) {
                    myresults3=myresults3+"  "+cbox2.getText().toString();
                }
                if(cbox3.isChecked()) {
                    myresults3=myresults3+"  "+cbox3.getText().toString();
                }
                if(cbox4.isChecked()) {
                    myresults3=myresults3+"  "+cbox4.getText().toString();
                }
                tvResult.setText(myresults1+myresults2+myresults3);
            }
    };
}
```

示例工程的运行结果如图 4-16 所示。

图 4-16　示例工程的运行结果

4.4　本 章 小 结

本章主要学习设计用户界面需要的基础知识。学习了常见的界面布局方式及实现方法，常用的布局方式有线性布局、表格布局、相对布局、绝对布局和框架布局，它们各有特点，适于不同的应用。本章还学习了 TextView、EditText、Button、CheckBox 和 RadioButton 等控件，以及常见事件的监听与处理方法。本章的重点是掌握常见布局方式的使用方法和理解 Android 中的事件处理机制，勤于练习是学好本章内容的关键。

习　　题

1．Android 应用程序的界面布局有哪几种方式？

2．什么情况下适合采用绝对布局方式？采用这种布局时要注意什么问题？

3．如果想让 TextView 中的文本居中显示，应当设置 android:gravity 属性的值还是设置 android:layout_ gravity 属性的值为 center？

4．分别以线性、相对布局的方式，实现一个 Activity 的界面。要求界面有说明文字，以及姓名、性别、年龄输入框，底部给出确定和取消两个按钮。

5．设计一个提交订单的用户界面，要求在不同屏幕尺寸时显示效果相同。

6．设计一个应用程序，处理 EditText 上触发的事件。当用户在 EditText 中输入文本时，在其下方的 TextView 中显示已经输入的文本长度。

7．设计一个如图 4-17 所示的用户界面。当用户输入用户名和年龄后，单击“回显”按钮则在下部回显出用户输入的用户名和年龄。



(a)



(b)

图 4-17　习题 7 的用户界面

8. 实现一个 Activity 的 UI 界面，要求显示一组单选按钮，选项分别是“普通用户”、“银卡用户”、“金卡用户”，当用户选择不同选项时，Activity 底部用文字显示出用户类型。

9. 实现一个 Activity 的 UI 界面，由用户选择喜欢的运动项目，要求显示一组复选框，选项分别是“篮球”、“足球”、“网球”、“游泳”、“慢跑”，当用户选择不同选项时，在 Activity 底部用文字显示出所有的选中项目。

10. 设计并实现一个 UI 界面，界面中包括一组单选按钮、一个文本输入框和一个“确定”按钮，单选按钮包括 3 个选项，分别是“普通用户”、“银卡用户”、“金卡用户”，要求用户选择一种用户类型，并在文本框中输入一个金额后，单击“确定”按钮。用户单击“确定”按钮后，在 Activity 底部用文字显示出折扣后的金额，3 种用户的折扣率分别为 0.9、0.8、0.7。

11. 设计一个加法计算器，如图 4-18 所示。用户在前两个文本框中输入整数，单击＝按钮，在第 3 个文本框中显示运算结果。



图 4-18　习题 11 的用户界面

第5章 常用 UI 界面控件

除了第 4 章介绍的 TextView、EditText、Button、CheckBox 和 RadioGroup 等控件以外，Android 系统还提供了很多其他界面控件，例如 Toast 信息提示、对话框、下拉列表、选项卡、日期时间控件和菜单等，以帮助用户完成丰富多彩的界面设计。本章介绍常用界面控件及其相关的设计和使用方法。了解这些控件的样式及使用方法对于用户界面设计是十分重要的。

5.1 信息提示和对话框

5.1.1 Toast

Toast 是一个 View 视图，用于快速为用户显示少量的信息。Toast 在应用程序上浮动显示信息给用户，它永远不会获得焦点且显示的时间有限，不会打断用户当前的操作，主要用于显示一些简短的帮助或提示信息。由于不会获得焦点，Toast 不接受任何用户的输入或交互。

创建 Toast 的一般步骤如下。

步骤 1：引入包文件：

```
import android.widget.Toast;
```

步骤 2：调用 Toast 的静态方法 makeText()或 make()，设置显示的文本和时长。makeText()方法的定义如下：

```
Toast.makeText(Context context,CharSequence text,int duration)
```

其中，参数 context 是当前的上下文环境。可通过调用 getApplicationContext()方法或直接使用 this 取得当前的上下文环境。参数 text 是要显示的字符串，也可以使用 R.string.××引用资源文件中的字符串 ID。参数 duration 是显示的时间长短。Toast 定义有两个符号常量 LENGTH_LONG 和 LENGTH_SHORT，分别指定长和短的显示时长。也可以直接使用毫秒数设置显示时长，例如 2000。

如果需要显示较为复杂的信息，可以调用 setView(view)方法，以添加 View 组件的方式来实现。另外，可以使用 setGravity()方法来定位 Toast 在屏幕上的位置，setGravity()方法的定义如下：

```
setGravity(int gravity, int xOffset, int yOffset)
```

其中，参数 gravity 用于设置 Toast 在屏幕中显示的位置，可以直接使用 Gravity 类定义的一些关于位置的符号常量，如 Gravity. TOP、Gravity. CENTER_VERTICAL 等；后两个参数是相对于第一个参数所设置位置的横向 X 轴和纵向 Y 轴的偏移量，正数向右、向下偏移，负数向左、向上偏移，如果设置的偏移量超过了屏幕的范围，Toast 将在屏幕内靠近超出的那个边界显示。

如果不设置 gravity 参数，Toast 将在默认位置显示，其位置为窗口下部，水平居中。

步骤 3：调用 Toast 的 show()方法显示提示信息。

【例 5-1】 示例工程 05_ToastExample 演示了 Toast 的各种使用方法。

Activity 中设置了 3 个按钮，单击按钮 Btn1，则使用 LENGTH_SHORT 参数显示短时的 Toast 提示；单击按钮 Btn2，则使用 LENGTH_LONG 参数显示长时的 Toast 提示；单击按钮 Btn3，则使用 setView(view)方法显示一个图片，并调用 setGravity()方法设置在窗口顶部居中的位置显示。Activity 中与此有关的部分代码如下：

```
Btn1.setOnClickListener(new Button.OnClickListener() {   //侦听按钮 1 被按下的动作
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "短时的 Toast 信息", Toast.LENGTH_SHORT).show();
    }
});

Btn2.setOnClickListener(new Button.OnClickListener() {   //侦听按钮 2 被按下的动作
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "长时的 Toast 信息", Toast. LENGTH_LONG).show();
    }
});

Btn3.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        Toast toast=new Toast(getApplicationContext());//实例化 Toast
        ImageView myview=new ImageView(getApplicationContext());
                                              //实例化 ImageView
        myview.setImageResource(R.drawable.image1);//和指定的图片关联
        toast.setView(myview);                //将 Toast 实例和图片实例关联
        toast.setGravity(Gravity.TOP, 0, 0);        //定位显示位置，否则以默认位置显示
        toast.show();                         //显示 Toast
    }
});
```

示例工程的运行结果如图 5-1 所示。图中为单击按钮 Btn2 后的提示信息。

5.1.2 状态栏提醒 Notification

与 Toast 不同，Notification 是显示在状态栏的提醒信息。同样，它也不会打断用户





图 5-1　示例工程 05_ToastExample 的运行结果

当前的操作，而且它支持更复杂的点击事件响应。

使用 NotificationManager 来管理 Notification。NotificationManager 负责“发出”与“取消”Notification。使用 Notification 需要导入 3 个包文件：

```
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
```

创建 Notification 的一般步骤如下。

步骤 1：得到一个 NotificationManager 的引用。通常通过调用 getSystemService (NOTIFICATION_SERVICE)方法来得到 NotificationManager 的引用，例如：

```
private NotificationManager mNotificationManager;
mNotificationManager= (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
```

步骤 2：创建一个 Notification 实例。其构造方法的定义如下：

```
Notification(int iconid, CharSequence text, long time)
```

其中，第一个参数是 icon 的资源 ID；第二个参数是设置在状态栏上展示的滚动信息；第三个参数是发通知的时间，也就是通知栏下拉后，左边显示的时间。一般用 System.currentTimeMillis()提取系统时间设置第三个参数。

以下是一个示例：

```
Notification myNotify= new Notification(R.drawable.image1,"这是 Notification 提示信息",System.
currentTimeMillis());
```

步骤 3：设置 Notification 的参数。当系统发出一个 Notification 时，仅在状态栏短时显示一个标题性的提醒文字，其余时间则显示一个提醒图标。按住状态栏向下滑动，则会

显示全部提醒信息。调用 setLatestEventInfo()方法,可以设置按住状态栏向下滑动时显示的提醒信息。该方法定义如下:

```
setLatestEventInfo(Context context, CharSequence contentTitle, CharSequence
contentText,PendingIntent contentIntent)
```

以下是一个设置参数的示例:

```
Context context=getApplicationContext();
CharSequence contentTitle="您好";
CharSequence contentText="这是给您的 Notification 提示";
Intent notifyIntent=new Intent(android.content.Intent.ACTION_VIEW);
PendingIntent intent=PendingIntent.getActivity(MainActivity.this, 0, notifyIntent, android.content.
Intent.FLAG_ACTIVITY_NEW_TASK);
myNotify.setLatestEventInfo(context, contentTitle, contentText, intent);
```

其中,PendingIntent 是 Intent 的包装,这里是启动 Intent 的描述。PendingIntent. getActivity()方法返回的 PendingIntent 表示此 PendingIntent 实例中的 Intent 是用于启动 Activity 的 Intent。PendingIntent. getActivity()的参数依次为 Context、发送者的请求码(可以填 0)、用于系统发送的 Intent、标志位。PendingIntent 将在第 7 章详细介绍。

步骤 4:调用 NotificationManager 的 notify()方法,显示指定的 Notification:

```
mNotificationManager.notify(SIMPLE_NOTFICATION_ID, myNotify);
```

调用 NotificationManager 的 cancel()方法,则取消指定的 Notification:

```
mNotificationManager.cancel(SIMPLE_NOTFICATION_ID);
```

【例 5-2】 工程 05_NotificationExample 演示了有关 Notification 提示的使用方法,MainActivity 类的主要代码如下:

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private NotificationManager mNotificationManager;
    private int SIMPLE_NOTFICATION_ID=0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mNotificationManager=(NotificationManager)getSystemService
        (NOTIFICATION_SERVICE);
                //得到一个 NotificationManager 的引用
        final Notification myNotify=new Notification(R.drawable.image1, "这是 Notification 提示信
        息",System.currentTimeMillis());
                //创建一个 Notification 实例
        Button btnstart=(Button)findViewById(R.id.btn_1);
        Button btncancel=(Button)findViewById(R.id.btn_2);
```

```
            Button btnfinish= (Button)findViewById(R.id.btn_3);
            final TextView txt= (TextView)findViewById(R.id.tv_message);

            btnstart.setOnClickListener(new OnClickListener() {
                    //"显示提示"按钮对应的单击事件
              public void onClick(View v) {
                Context context= getApplicationContext();
                CharSequence contentTitle= "您好";
                CharSequence contentText= "这是给您的 Notification 提示";
                txt.setText("Notification 示例----您选中了显示提示\n");
                Intent notifyIntent=new Intent(android.content.Intent.ACTION_VIEW);
                PendingIntent intent= PendingIntent.getActivity(MainActivity.this,
                    0, notifyIntent, android.content.Intent.FLAG_ACTIVITY_NEW_TASK);
                myNotify.setLatestEventInfo(context, contentTitle, contentText, intent);
                            //设置 Notification 的参数
                mNotificationManager.notify(SIMPLE_NOTFICATION_ID, myNotify);
                            //调用 notify()方法,显示指定的 Notification
                }
            });
            btncancel.setOnClickListener(new OnClickListener() {
                    //"取消提示"按钮对应的单击事件
              public void onClick(View v) {
                txt.setText("Notification 示例----您取消了显示提示\n");
                mNotificationManager.cancel(SIMPLE_NOTFICATION_ID);
              }
          });
            btnfinish.setOnClickListener(new OnClickListener() {
                    //"结束程序"按钮对应的单击事件
              public void onClick(View v) {
                 finish();//退出应用程序
            }
          });
        }
    }
```

程序运行后,单击“显示提示”按钮,状态栏就会显示消息提示,运行结果如图 5-2 所示。

按住状态栏,向下滑动,则会显示全部提醒信息,如图 5-3 所示。



图 5-2 示例工程 05_NotificationExample 的运行结果



图 5-3 下拉状态栏后的运行结果

5.1.3 带自动输入提示的文本框 AutoCompleteTextView

Android 系统中提供了两种类型的智能文本输入框，即 AutoCompleteTextView 和 MultiAutoCompleteTextView。在输入框中输入部分内容后，和内容相关的选项自动被列出来，用户可以从中选择一项，从而简化输入过程。本节介绍 AutoCompleteTextView 的使用方法。

AutoCompleteTextView 继承自 android. widget. EditText，在 android. widget 包中。输入框显示的自动提示文本一般是从一个数据适配器中获取，所以使用 AutoCompleteTextView 需用 import 语句引入 android. widget. AutoCompleteTextView 和 android. widget. ArrayAdapter。实现这些控件的关键是设置列表资源的适配器(Adapter)，主要步骤如下。

步骤 1：在 XML 布局文件中加入 AutoCompleteTextView 控件。

```
<AutoCompleteTextView
    android:id="@+id/auto_complete"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

在 Activity 中获得 AutoCompleteTextView 实例，例如：

```
atv= (AutoCompleteTextView)findViewById(R.id.auto_complete);
```

步骤 2：声明字符串数组，数组中的字符串就是将来自动提示的字符串，例如：

```
String[] strs= {"abc","abcd","bcd","bcde"};
```

步骤 3：利用字符串数组创建并实例化适配器，例如：

```
ArrayAdapter<String> adapter= new ArrayAdapter<String> (
        this,android.R.layout.simple_dropdown_item_1line, strs);
```

其中，第一个参数是当前的上下文环境；第二个参数是资源 ID，此处为下拉列表的 XML 布局文件 ID；第三个参数是字符串数组。

系统会根据用户在输入框中已经输入的文字到适配器中查找前几个字符与输入相匹配的字符串，并将其列于输入框的下方，供用户选择。

步骤 4：为 AutoCompleteTextView 控件设置适配器，例如：

```
atv.setAdapter(adapter);
```

【例 5-3】 工程 05_AutoCompleteTextViewExample 演示了具有自动提示功能的 AutoCompleteTextView 控件的用法。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    static final String[] AOTUSTR= new String[] {
```

```
        "北京站","北京火车站", "北京西站", "北京天安门", "北京天气", "北京天南小吃店" };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter= new ArrayAdapter<String>(
                this,android.R.layout.simple_dropdown_item_1line, AOTUSTR);
                //创建适配器
        AutoCompleteTextView textView= (AutoCompleteTextView)findViewById(
                R.id.auto_complete);
        textView.setAdapter(adapter);
                //为 AutoCompleteTextView 控件设置适配器
        }
}
```

可以将字符串数组定义在 XML 资源文件中，例如，在 arrays.xml 资源文件中定义字符串：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="aotuStrings">
        <item>北京站</item>
        <item>北京火车站</item>
        <item>北京西站</item>
        <item>北京天安门</item>
        <item>北京天气</item>
        <item>北京天南小吃店</item>
    </string-array>
</resources>
```

在 Java 程序中通过 getResources().getStringArray(R.array.字符串数组名称)的方式引用上述字符串。则创建并实例化适配器的语句如下：

```
ArrayAdapter<String> adapter= new ArrayAdapter<String>
(this,android.R.layout.simple_dropdown_item_1line,
getResources().getStringArray(R.array.aotuStrings));
```

示例工程 05_AutoCompleteTextViewExample 的运行结果如图 5-4 所示。当输入两个字符之后就会根据当前已经输入的文字列出自动提示。

5.1.4 提示对话框 AlterDialog

在 Android 中，对话框是一种显示于 Activity 上的界面元素，当显示对话框时，当前 Activity 失去焦点而由对话框负责所有的交互。一般来说，对话框用于给出提示信息或弹出一个与主进程直接相关的子程序。常见的对话框有提示对话框 AlterDialog、进度对

图 5-4　示例工程 05_AutoCompleteTextViewExample 的运行结果

话框 ProcessDialog、日期选择对话框 DatePickerDialog 和时间选择对话框 TimePickerDialog 等。

当对话框第一次被显示时，在程序中通过回调方法 onCreateDialog()来完成对话框实例的创建。该方法需要传入代表对话框的 ID 参数。之后不再重复创建该实例，如果需要再次显示对话框，只需调用 showDialog()方法传入对话框的 ID 来显示指定的对话框。每次对话框被显示之前都会调用 onPrepareDialog()方法，所以如果不重写该方法，每次显示的对话框都是最初创建的那个。关闭对话框时可以通过调用 Dialog 类的 dismiss()方法来实现，但通过这种方法关闭的对话框并不会彻底销毁，Android 会在后台保留其状态，因此可以为对话框设置 onDismissListener()的监听，并重写其中的 onDismiss()方法来解决这一问题。如果需要让对话框在关闭之前彻底被清除，可以调用 removeDialog()方法并传入 Dialog 的 ID 值来彻底释放对话框资源。

通常调用 showDialog(int)方法显示对话框，调用 dismissDialog(int)方法关闭对话框。另外也可以直接调用 Dialog 的 dismiss()和 cancel()方法。当用户单击“返回”按钮时，对话框也会被取消(cancel)。所以如果想监听对话框何时被取消时，可以实现 DialogInterface. OnDismissListener 接口，然后调用 setOnDismissListener()方法去监听。

本节介绍对话框中最常用、也是最简单的 AlterDialog 的设计方法。AlterDialog 是一个消息提示对话框，能构造默认的 3 个按钮，分别用于“是”、“否”和“取消”。AlterDialog 继承自 android. app. Dialog 类，在 android. app 包中，创建 AlertDialog 对话框的主要步骤如下。

步骤 1：获得 AlertDialog 的静态内部类 Builder 对象，并由该类来创建对话框。

步骤 2：通过 Builder 对象设置对话框的属性。

可通过调用 setIcon()方法设置显示在对话框标题左侧的图标，调用 setTitle()方法设置对话框的标题，调用 setMessage()方法设置对话框中显示的文字信息，调用 setView()方法设置对话框中显示的内容。其中 setView()方法的参数为一个 View 实例名，调用该方法可

以在对话框中显示 Widget 控件，例如：

```
.setIcon(R.drawable.icon)
.setTitle("用户登录界面")
.setView(dialogshow)                          //参数为一个 View 实例名
```

步骤 3：设置对话框的按钮以及单击按钮将要响应的事件。

对话框中可以有“是”、“否”和“取消”3 个按钮，生成器 Builder 负责设置对话框上的按钮，并为按钮注册 DialogInterface. OnClickListener 监听器。DialogInterface. OnClickListener 在 android. content 包中，事件处理方法是 onClick()方法。无论用户单击哪一个按钮，对话框都会消失，并导致接口中的 onClick()方法被调用。onClick()方法的定义如下：

```
abstract void onClick(DialogInterface dialog, int which)
```

其中，参数 dialog 就是当前要消失的对话框，参数 which 是用户单击的按钮，其取值可以是 DialogInterface. BUTTON_NEGATIVE、DialogInterface. BUTTON_NEUTRAL、DialogInterface. BUTTON_POSITIVE。

步骤 4：调用 Builder 对象的 create()方法可以返回它所构建的对话框，例如：

```
AlertDialog dialog=builder.create();
```

步骤 5：调用 show()方法显示对话框，调用 hide()方法可以隐藏对话框。

如果不希望用户单击设备的“返回”按钮使对话框消失，而是要求用户必须单击对话框中的按钮，则可以通过调用 setCancelable(false)方法来进行设置。

【例 5-4】　工程 05_AlertDialogExample 演示了 AlterDialog 的用法。

当单击按钮后，弹出 AlertDialog 对话框。MainActivity 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        btnstart.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                AlertDialog.Builder myDialog=new AlertDialog.Builder(
                        MainActivity.this);                //创建 AlertDialog.Builder 对象
                myDialog.setTitle("提示");                          //设置标题
                myDialog.setMessage("这是一个 AlertDialog 对话框!");  //设置显示消息
                myDialog.setNegativeButton("取消", null);
                myDialog.setPositiveButton("确定", null);
                myDialog.show();                                   //显示对话框
            }
```

```
            });
        }
    }
```

示例工程 05_AlertDialogExample 的运行结果如图 5-5 所示。



图 5-5　示例工程 05_AlertDialogExample 的运行结果

除了按钮对话框，还可以创建列表、单选和多选对话框。通过调用 setItems()方法，可以在 AlertDialog 中添加列表项；通过调用 setSingleChoiceItems()/setMultiChoiceItems()方法，可以在 AlertDialog 中添加单选/多选按钮，其余语法与创建普通的按钮对话框类似。

5.1.5　进度条对话框 ProcessDialog

ProgressDialog 是 AlertDialog 的子类，在 android.app 包中。除了 AlertDialog 功能外，它还能显示进度圈或进度条。当进行一个比较耗时的操作时，弹出一个 ProgressDialog 可以使界面更友好。

可以直接通过 new 的方式来创建一个 ProgressDialog，通过调用 setProgressStyle()方法设置进度条的样式。进度条有两种样式：一种是水平进度条，另一种是圆圈进度条。创建进度条后，一般会启动另外一个线程调用 incrementProgressBy()方法来设置进度。

ProgressDialog 可以显示一个文本 message 或者是一个自定义的 View，但是要注意，文本 message 和自定义的 View 只能存在一个。

【例 5-5】　示例工程 05_ProcessDialogExample，演示两种样式的进度条对话框。

```
//package 和 import 语句略
public class ProcessDialogActivity extends Activity {
    private Button button1,button2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
            button1= (Button)findViewById(R.id.btn_1);
            button2= (Button)findViewById(R.id.btn_2);
            button1.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View v) {
                  ProgressDialog progressDialog= new ProgressDialog(ProcessDialog-Activity.this);
                    //实例化一个 ProgressDialog
                  progressDialog.setTitle("提示信息");
                  progressDialog.setMessage("正在下载中,请稍候……");
                  progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
                    //设置 ProgressDialog 的显示样式,STYLE_SPINNER 代表的是圆圈进度条
                  progressDialog.show();
                    //显示进度对话框
                }
            });
            button2.setOnClickListener(new Button.OnClickListener() {
                public void onClick(View v) {
                  ProgressDialog progressDialog= new ProgressDialog(ProcessDialog-Activity.this);
                   //实例化一个 ProgressDialog
                  progressDialog.setTitle("提示信息");
                  progressDialog.setMessage("正在下载中,请稍候……");
                    //设置最大进度,ProgressDialog 的进度范围是 1~10000
                  progressDialog.setMax(100);
                  progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                    //设置 ProgressDialog 的显示样式, STYLE_HORIZONTAL 代表的是水平进度条
                  progressDialog.show();
                    //显示进度对话框
                }
            });
        }
    }
```

本例程序中,单击屏幕其他部分,进度条对话框就会消失。如果希望其不消失可以调用 progressDialog. setCancelable(false)方法,这样对话框就不能被取消。两种进度条的运行结果如图 5-6 所示。

5.2 常用容器类控件

5.2.1 列表控件 ListView

ListView 是 android. view. GroupView 的间接子类。ListView 以列表的形式展示内容,可以按设定的规则自动填充并展示一组列表信息,并且能够根据数据的长度自适应显示。如果显示内容过多,会出现垂直滚动条。ListView 能够通过适配器将数据和自身绑定,在有限的屏幕上提供大量内容供用户选择。



(a)



(b)

图 5-6　示例工程 05_ProcessDialogExample 的运行结果

ListView 的显示需要 3 个要素：用来展示列表的 View(ListVeiw)；用来把数据映射到 ListView 上的适配器；具体被映射到列表中的字符串、图片，或者基本组件。

列表的适配器有 3 种类型，分别是 ArrayAdapter、SimpleAdapter 和 SimpleCursorAdapter，其中 ArrayAdapter 最常用，也最简单，列表内容为字符串。SimpleAdapter 有最好的扩充性，可以自定义出各种效果。SimpleCursorAdapter 可以看作是 SimpleAdapter 与数据库的简单结合，可以把数据库的内容以列表的形式展示出来。ListView 和上述适配器都在 android.widget 包中。

本节以 ArrayAdapter 为例介绍 ListView 的使用方法。ListView 支持单击事件的处理，包括列表项 Item 被单击和被选择。Item 被单击会触发 OnItemClick 事件，Item 被选择会触发 OnItemSelected 事件。响应上述事件会调用 onItemClick()或 onItemSelected()方法，这两个方法都有 4 个参数：AdapterView<?> parent、View view、int position、long id，其中 parent 表示适配器控件，就是发生事件的 ListView；view 表示适配器内部的控件，就是是被单击或选择的 Item 子项；position 表示子项的位置；id 表示子项的 ID 号。

【例 5-6】 示例工程 05_ListViewExample 演示了如何设置 ListView 及对 Item 被单击和选择的事件作出响应。

实现过程如下。

步骤 1：将列表项文字定义到一个字符数组中。可以直接在 Java 文件中定义，也可以定义到 XML 资源文件中，然后在 Java 文件中引用。

步骤 2：定义列表项的布局。可以直接使用 Android 系统提供的布局文件，例如，android.R.layout.simple_list_item_1，这样就不需要自己定义列表项的布局。也可以定制自己的布局，这时需要新建一个 XML 布局文件，定义列表中每一行的布局。本例中自定义了布局文件，名为 list_item.xml，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:textSize="18sp"
    android:textColor="#0000ff">
</TextView>
```

步骤 3：定义 Activity 使用的 XML 布局文件 activity_main.xml，其中必须包含一个 ListView 控件。

activity_main.xml 文件的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
  <TextView
        android:id="@+id/tv_message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="ListView示例\n"
        />
  <ListView
       android:id="@+id/listview"
       android:background="#cccccc"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent" />
</LinearLayout>
```

除了与其他 Widget 控件类似的属性外，ListView 的常用属性还有 android:divider 和 android:dividerHeight，前者用于设置相邻两个列表项之间的分界线样式，后者用于设置相邻两个列表项之间的分界线高度。本例中这些属性都使用默认值。

步骤 4：定义 Activity，在 Activity 中实例化 ListView 控件，绑定 ListView 控件的数据源，处理列表项的单击或选择事件等。对应的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private String[] listItem=new String[]{"北京","天津","上海","河北","河南"};

    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       ListView mylistview= (ListView)findViewById(R.id.listview);
                          //实例化 ListView 控件
       ArrayAdapter<String> adapter=new ArrayAdapter<String> (this,
```

```
                R.layout.list_item, listItem);
        mylistview.setAdapter(adapter);
                        //绑定 ListView 控件的数据源
        mylistview.setOnItemClickListener(new AdapterView.OnItemClickListener()
         {                                              //处理列表项的单击事件
            public void onItemClick(AdapterView<?>parent, View view, int position, long id) {
                String itemString= ((TextView)view).getText().toString();
                        //获取单击项的文字
                Toast.makeText(MainActivity.this, "您单击了列表项:"+ itemString, Toast.LENGTH_
                LONG).show();
            }
         });
        mylistview.setOnItemSelectedListener(new OnItemSelectedListener() {
            //处理列表项的选择事件
            public void onItemSelected(AdapterView<?>parent, View view, int position, long id) {
                Toast.makeText(MainActivity.this, "您选择了列表项:"+ ((TextView)view).getText
                ().toString(), Toast.LENGTH_LONG).show();
            }
         });
    }
}
```

示例工程 05_ListViewExample 的运行结果如图 5-7 所示。



图 5-7 示例工程的运行结果

除了使用前述方法,实现 ListView 的 Activity 还可以通过继承 ListActivity 类实现。ListActivity 是 Activity 的子类,是 ListView 和 Activity 的结合。ListActivity 不需要调用 setContentView()方法来设定布局,它有一个默认的布局,由一个位于屏幕中心的全屏列表构成。

在 ListActivity 中,如果不想使用默认的布局,可以在 onCreate()方法中通过 setContentView()方法设定自己的布局。如果指定自己定制的布局,布局文件中必须包含一个 id 为"@id/android:list"的 ListView。若还指定了一个 id 为"@id/android:empty"的控件,则当 ListView 中没有数据要显示时,这个控件就会被显示,同时 ListView 会被隐藏。

5.2.2 下拉列表 Spinner

Spinner 也是 android. view. GroupView 的间接子类。Spinner 是一个能从多个选项中选择一个选项的控件,它使用浮动菜单为用户提供选择。Spinner 的用法有很多地方与 ListView 控件类似,不再赘述。

【例 5-7】 示例工程 05_SpinnerExample 演示了对 Spinner 的用法。

实现过程如下。

步骤 1：将列表项文字定义到一个字符数组中。与 ListView 类似，字符数组可以直接在 Java 文件中定义，也可以定义到 XML 资源文件中，然后在 Java 文件中引用。

步骤 2：定义列表项的布局。与 ListView 类似，可以直接使用 Android 系统提供的布局文件，例如，android. R. layout. simple_list_item_1、android. R. layout. simple_spinner_item，这样就不需要自己定义列表项的布局。也可以定制自己的布局，这时需要新建一个 XML 布局文件，定义列表中每一行的布局。本节示例工程中使用与 5.2.1 节相同的布局文件 list_item. xml。

步骤 3：定义 Activity 使用的 XML 布局文件 activity_main. xml，其中必须包含一个 Spinner 控件，定义 Spinner 控件的部分代码如下。

```
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="下拉列表示例" />
```

步骤 4：定义 Activity，在 Activity 中实例化 Spinner 控件，绑定 Spinner 控件的数据源，处理相关事件等。

和按钮等大多数控件一样，Spinner 控件也是通过创建一个监听器来监听用户的动作的。当用户选择其中一个选项时，便会触发响应。

下拉列表的事件监听有多种，常用的有 3 种。

(1) 当列表项被单击时所触发的事件：

```
setOnItemClickListener(AdapterView.OnItemClickListener listener);
```

(2) 当列表项被选择时所触发的事件：

```
setOnItemSelectedListener(AdapterView.OnItemSelectedListener listener);
```

(3) 当列表项被长时间按住时所触发的事件：

```
setOnItemLongClickListener(AdapterView.OnItemLongClickListener listener);
```

本例 MainActivity 类的部分代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Spinner myspinner= (Spinner)findViewById(R.id.spinner1);
        ArrayAdapter<String> adapter = new ArrayAdapter<String> (this, R.layout.list_item,
        getResources().getStringArray(R.array.listStrings));
        myspinner.setAdapter(adapter);
```

```
            myspinner.setOnItemSelectedListener(new OnItemSelectedListener() {
                public void onItemSelected(AdapterView<?>parent, View view, int position, long id) {
                    Spinner spinner= (Spinner)parent;
                    Toast.makeText(MainActivity.this, "您选择了列表项:"+ spinner.getSelectedItem
                    ().toString(), Toast.LENGTH_LONG).show();
                }
                public void onNothingSelected(AdapterView<?>parent) {
                    }
            });
        }
}
```

示例工程 05_SpinnerExample 的运行结果如图 5-8 所示。与 ListView 不同的是，当用户单击控件时，下拉列表才会显示，当用户单击某个列表项后，系统会响应相应的事件，同时列表项会自动收回。



图 5-8　示例工程 05_SpinnerExample 的运行结果

5.2.3　选项卡 TabHost

TabHost 也是 android.view.GroupView 的间接子类。TabHost 是一个用来存放多个 Tab 标签页的容器，每一个 Tab 页都可以使用自己的布局。TabHost 包含两个子对象：用户可以选择指定 Tab 页的标签和用来显示 Tab 页内容的 FrameLayout。使用 Tab 标签，可以通过多个 Tab 页切换显示不同的内容，这样在有限的手机屏幕中就可以显示尽可能多的信息。

实现 TabHost 时，一般通过 TabActivity 类的 getTabHost()方法取得 TabActivity 对应的 TabHost 对象，然后通过 addTab()方法来向 TabHost 中添加 Tab。每个 Tab 有 Tag(用来区分某个 Tab 页面)、Indicator(页标题，可以使用文字、图标等来显示)和 Content(页面内

容)等属性。其中 Content 可以使用 View 资源 ID、TabHost. TabContentFactory 实例或可以启动某个 Activity 的 Intent 实例。

每个 Tab 在切换时都会产生一个事件,要捕捉这个事件需要设置事件监听 setOnTabChangedListener()。

在 Android 中,通常使用 TabActivity 来制作 Tabs,所以显示 TabHost 的 Activity 通常都继承自 android. app. TabActivity。在 Java 程序中通常需要使用 import 语句引入 android. widget. TabHost 和 android. app. TabActivity。

TabActivity 的部分方法及其含义如下。

(1) public TabHost getTabHost():获得当前 TabActivity 的 TabHost。

(2) public TabWidget getTabWidget():获得当前 TabActivity 的 TabWidget。

(3) public void setDefaultTab(String tag):设置默认的 Tab。

Tab 的载体是 TabHost,需要调用 TabActivity. getTabHost()方法获取。以下是 TabHost 类的一些常用方法。

(1) public void addTab(TabHost. TabSpec tabSpec):添加 tab,参数 TabHost. TabSpec 通过下面的函数返回得到。

(2) public TabHost. TabSpec newTabSpec(String tag):创建 TabHost. TabSpec。

(3) public void clearAllTabs():清除所有的 Tabs。

(4) public void setCurrentTab(int index):使用索引值设置当前显示的 Tab。

(5) public void setCurrentTabByTag(String tag):使用 Tag 标志设置当前显示的 Tab。

【例 5-8】 示例工程 05_TabHostExample 演示了 TabHost 的用法。工程中定义了 3 个不同的 Tab 页,在每个 Tab 页中,分别显示 XML 布局文件中不同的内容。单击标签可切换显示不同的 Tab 页。

示例程序中到了 LayoutInflater 类,它的作用类似于 findViewById()。不同点是 LayoutInflater 是用来加载 res/layout/下的 XML 布局文件,并且实例化;而 findViewById()是加载 XML 布局文件中的某个的具体 Widget 控件(如 Button、TextView 等)。

本例采用 View 资源 ID 的方法来设置 Tab 的 Content。

```
//package 和 import 语句略
public class MainActivity extends TabActivity {
                //继承自 TabActivity,而非 Activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TabHost tabHost=getTabHost();
                //调用 TabActivity 的 getTabHost()方法获得 TabHost 对象
        LayoutInflater.from(this)
            .inflate(R.layout.activity_main,tabHost.getTabContentView(),true);
        tabHost.addTab(tabHost.newTabSpec("tab1")
```

```
                    .setIndicator("TAB-1")
                    .setContent(R.id.txt1));
                    //通过 TabHost 创建 Tab 选项
            tabHost.addTab(tabHost.newTabSpec("tab2")
                    .setIndicator("TAB-2")
                    .setContent(R.id.edit2));
            tabHost.addTab(tabHost.newTabSpec("tab3")
                    .setIndicator("TAB-3")
                    .setContent(R.id.edit3));
    }
}
```

示例工程 05_TabHostExample 的运行结果如图 5-9 所示。



图 5-9　示例工程 05_TabHostExample 的运行结果

提示： Android 新版本不建议使用 TabActivity，而是使用 Fragment。有关详细用法请读者参阅相关 API 文档。

5.3　日期和时间控件

Android 系统提供了一些与日期和时间有关的控件。其中，DatePicker 用来实现日期输入设置，TimePicker 用来实现时间输入设置。DatePickerDialog 用来显示日期对话框，TimePickerDialog 用来显示时间对话框。AnalogClock 用来显示一个指针式时钟，DigitalClock 用来显示一个数字式时钟。

5.3.1　DatePicker 和 TimePicker

1. DatePicker

DatePicker 继承自 android. widget. FrameLayout 类，在 android. widget 包中。在 Android 中，DatePicker 用来实现日期输入设置，日期的设置范围为 1900 年 1 月 1 日至 2100 年 12 月 31 日。改变日期会触发 onDateChanged 事件，所以要监听日期值的改变，需要实现接口 android. widget. DatePicker. OnDateChangedListener 中的 onDateChanged() 方法。

DatePicker 的常用方法有以下一些。

(1) getDayOfMonth()：获取当前日期的日。

(2) getMonth()：获取当前日期的月。

(3) getYear()：获取当前日期的年。

(4) init (int year, int monthOfYear, int dayOfMonth, DatePicker.OnDateChangedListener onDateChangedListener)：初始化日期，并注册日期值改变事件的监听器。

(5) updateDate(int year,int month,int dayOfMonth)；更新当前日期。

2. TimePicker

TimePicker 也继承自 android.widget.FrameLayout 类，在 android.widget 包中。TimePicker 向用户显示一天中的时间，并允许用户进行选择。时间的改变会触发 OnTimeChanged 事件，所以要监听时间值的改变，需要实现接口 android.widget.TimePicker.OnTimeChangedListener 中的 onTimeChanged()方法

TimePicker 常用的方法有 5 个。

(1) setCurrentMinute(Integer currentMinute)：设置当前时间的分钟。

(2) getCurrentMinute()：获取当前时间的分钟。

(3) setCurrentHour(Integer currentHour)：设置当前时间的小时

(4) getCurrentHour()：获取当前时间的小时。

(5) setIs24HourView(true)：设置为 24 小时制显示。

【例 5-9】 示例工程 05_DateAndTimePickerExample 演示了 DatePicker 和 TimePicker 控件的用法。

程序主界面设置了两个 Button 按钮，单击第一个按钮，跳转到 DatePickerActivity，显示 DatePicker 控件；单击第二个按钮，跳转到 TimePickerActivity，显示 TimePicker 控件。

DatePickerActivity 类的主要代码如下：

```
//package 和 import 语句略
public class DatePickerActivity extends Activity {
    private DatePicker mydatePicker;
    private TextView textDate;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.date);
        this.setTitle("日期控件的示例");
        textDate= (TextView) findViewById(R.id.textDate);
        mydatePicker= (DatePicker)findViewById(R.id.datePicker);
                //获取 DatePicker 对象
        Calendar calendar=Calendar.getInstance(Locale.CHINA);
        int year        =calendar.get(Calendar.YEAR);
        int monthOfYear=calendar.get(Calendar.MONTH);
```

```
            int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
            mydatePicker.init(year, monthOfYear, dayOfMonth, new OnDateChanged-Listener() {
                    //初始化日期,并注册日期值改变事件的监听器
                public void onDateChanged(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
                    textDate.setText("\n您选择的日期是: "+ year+ "年"+ (monthOfYear+ 1) + "月"+
                   dayOfMonth+ "日");
                }
            });
        }
    }
```

TimePickerActivity 类的主要代码如下:

```
//package 和 import 语句略
public class TimePickerActivity extends Activity {
        private TimePicker mytimePicker;
        private TextView textTime;
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.time);
            this.setTitle("时间控件的示例");
            textTime = (TextView) findViewById(R.id.textTime);
            mytimePicker= (TimePicker)findViewById(R.id.timePicker);
                    //获取 TimePicker 对象
            Calendar calendar= Calendar.getInstance(Locale.CHINA);
            int hour = calendar.get(Calendar.HOUR_OF_DAY);
            int minute = calendar.get(Calendar.MINUTE);
            mytimePicker.setCurrentHour(hour);
                    //设置当前的小时
            mytimePicker.setCurrentMinute(minute);
                    //设置当前的分钟
            mytimePicker.setOnTimeChangedListener(new OnTimeChangedListener()  {
                    //注册时间值改变事件的监听器
                public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {
                    textTime.setText("您选择的时间是: "+ hourOfDay+ "时"+ minute+
                    "分");
               }
            });
        }
    }
```

示例程序中,DatePicker 和 TimePicker 控件的使用方法类似,在布局中设置了 TextView 控件,当用户选择了日期或时间后,触发相应事件,在 TextView 中显示这个选择的日期或时间。运行结果如图 5-10 所示。



(a)



(b)

图 5-10　示例工程 05_DateAndTimePickerExample 的运行结果

5.3.2　DatePickerDialog 和 TimePickerDialog

DatePickerDialog 和 TimePickerDialog 都是 AlertDialog 的子类，是分别能让用户选择日期和时间的对话框，都在 android. app 包中。可以在程序中直接通过 new 的方式实例化 DatePickerDialog 类和 TimePickerDialog 类来得到一个日期、时间选择对话框，两者的使用方法非常类似。

1. DatePickerDialog

在 Java 程序中使用 DatePickerDialog 之前，需要用 import 语句引入 android. app. DatePickerDialog 和 android. widget. DatePicker。

对于 DatePickerDialog，其常用的构造方法定义如下：

```
DatePickerDialog (Context context, OnDateSetListener callBack, int year, int monthOfYear, int
dayOfMonth)
```

其中，第二个参数可以是一个 DatePickerDialog. OnDateSetListener 匿名内部类，当用户选择好日期后单击“完成”时，会调用里面的 onDateSet()方法。最后 3 个参数分别用于指定对话框弹出时默认选择的年、月、日。

如果要监听对话框中年、月、日值的改变，需要在 DatePickerDialog 控件中实现接口 android. widget. DatePicker. OnDateChangedListener 中的 onDateChanged()方法；如果监听对话框中确定按钮被按下，需要实现接口 onDateChangedListener 中的 onDateSet()方法。

2. TimePickerDialog

与 DatePickerDialog 类似，可以在程序中直接通过 new 的方式实例化 TimePickerDialog

类来得到一个时间选择对话框，在 Java 程序中使用 DatePickerDialog 之前需要用 import 语句引入 android. app. TimePickerDialog 和 android. widget. TimePicker。

DatePickerDialog 类常用的构造方法定义如下：

```
TimePickerDialog(Context context, OnTimeSetListener callBack, int hourOfDay, int minute, boolean
is24HourView)
```

其中，第二个参数可以是一个 TimePickerDialog. OnTimeSetListener 匿名内部类，当用户选择好时间后单击“完成”时，会调用里面的 onTimeset()方法。第 3 个参数(hourOfDay)和第 4 个参数(minute)为弹出的“时间”对话框的初始显示的小时和分钟，最后一个参数设置是否以 24 时制显示时间。

如果要监听对话框中时间值的改变，需要在 TimePickerDialog 控件中实现接口 android. widget. TimePicker. OnTimeChangedListener 中的 onTimeChanged()方法；如果监听对话框中确定按钮被按下，需要实现 TimePickerDialog. OnTimeSetListener 接口，并实现该接口中的 onTimeSet()方法。

【例 5-10】 示例工程 05_DateAndTimePickerDialogExample。

在主界面的布局中定义两个 Button 控件，单击第 1 个按钮，则弹出“日期选择”对话框，单击“完成”按钮关闭对话框后，利用 Toast 显示所选择的的日期。使用 new 运算符实例化 DatePickerDialog 时，第 2 个参数是一个 DatePickerDialog. OnDateSetListener 匿名内部类，当用户选择好日期单击“完成”按钮，会调用里面的 onDateSet()方法。

单击第 2 个按钮，弹出“时间选择”对话框，单击“完成”按钮关闭对话框后，利用 Toast 显示所选择的的时间。实例化 TimePickerDialog 的对象时，第 2 个参数是一个 TimePickerDialog. OnTimeSetListener 匿名内部类，当用户选择好时间后单击“完成”按钮，会调用里面的 onTimeset()方法。如果想要弹出对话框后，默认显示系统当前时间，可以利用 java. util. Calendar 类实现。

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.setTitle("日期和时间选择对话框示例");

        Button btn1= (Button)findViewById(R.id.btn_1);
        btn1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                //创建一个 DatePickerDialog
                DatePickerDialog datePickerDialog= new DatePickerDialog(MainActivity.this, new
                DatePickerDialog.OnDateSetListener() {
                    public void onDateSet(DatePicker view, int year, int monthOfYear, int
                    dayOfMonth) {
```




```
                    Toast.makeText (getApplicationContext (), "日期:" + year +  "-" +
                    (monthOfYear+ 1)+ "- "+ dayOfMonth, Toast.LENGTH_SHORT).show();
                }
            }, 2014, 9, 7);
            datePickerDialog.show();                //显示 DatePickerDialog
        }
    });

    Button btn2= (Button)findViewById(R.id.btn_2);
    btn2.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            //创建一个 TimePickerDialog
            TimePickerDialog timePickerDialog= new TimePickerDialog (MainActivity.this, new
            TimePickerDialog.OnTimeSetListener() {
                public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
                    Toast.makeText (getApplicationContext (), "Time: "+  hourOfDay+":"+
                    minute, Toast.LENGTH_SHORT).show();
                }
            }, 8, 15, true);
            timePickerDialog.show();                //显示 TimePickerDialog
        }
    });
  }
}
```

单击主界面的按钮后，分别弹出“日期”和“时间”对话框，如图 5-11 所示。



(a)



(b)

图 5-11　示例工程 05_DateAndTimePickerDialogExample 的运行结果

5.3.3 AnalogClock 和 DigitalClock

时钟控件包括 AnalogClock 和 DigitalClock,它们的功能都是显示时钟。AnalogClock 用来显示一个只有时针和分针的指针式时钟,DigitalClock 用来显示一个数字式时钟。

AnalogClock 类继承自 android.view.View,DigitalClock 类继承自 android.widge.TextView。它们都在 android.widge 包中。

【例 5-11】 示例工程 05_ClockExample 分别使用 AnalogClock 和 DigitalClock 控件在界面中显示了时钟。

XML 布局文件如下:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="数字时钟:"  />
    <DigitalClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="\n模拟时钟:"  />
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在 Activity 中直接引用上述布局文件,其运行结果如图 5-12 所示。

5.4 菜　　单

菜单是许多应用程序不可或缺的一部分,它能够在不占用界面空间的前提下为应用程序提供统一的功能和设置界面。Android 的菜单在正常情况下都是隐藏的,其主要目的是节省显示空间,而在需要调用它时按下移动设备上的 Menu 键就可以弹出来。但其对应的功能是需要程序设计者来实现的。如果在应用程序开发中没有实现菜单的功能,则在程序运行时按下移动设备上的 Menu 键不会显示任何菜单。

Android SDK 提供的菜单主要有如下 3 种。





图 5-12　数字时钟和模拟时钟

(1) 选项菜单(Options Menu)：最常规的菜单，即按 Menu 键时打开的菜单。

(2) 子菜单(SubMenu)：单击菜单项将弹出悬浮窗口显示子菜单项。子菜单不支持嵌套，即子菜单中不能再包括其他子菜单。

(3) 上下文菜单(Context Menu)：长按视图控件后出现的菜单。类似于 Windows 应用程序中的右键快捷菜单。

编写程序时一般不需要创建 Menu，每个 Activity 默认都包含一个 Menu 对象，一个 Menu 对象代表一个菜单，在 Menu 对象中可以添加菜单项 MenuItem，也可以添加子菜单 SubMenu。编程者只需添加菜单项和响应菜单项的单击事件，所以编写菜单程序一般包括创建和初始化菜单项及菜单项事件处理两个步骤。

5.4.1　选项菜单 Options Menu

Activity 中提供了两个回调方法 OnCreateOptionsMenu()和 OnOptionsMenuSelected()，用于创建菜单项和响应菜单项的单击。

1. 创建菜单项

public boolean onCreateOptionsMenu(Menu menu)方法的功能是初始化选项菜单。当按下移动设备上的 Menu 键时，Android 系统调用此方法生成一个菜单。可在此方法中添加指定的菜单项。但要注意：该方法只在首次显示菜单时调用一次，如果需要动态显示菜单项，则需要使用 onPrepareOptionsMenu()方法。

public boolean onPrepareOptionsMenu(menu)方法的功能是为程序准备选项菜单，在每次选项菜单显示前会调用该方法。可以通过该方法设置某些菜单项可用、不可用、动态显示菜单项的内容等。重写该方法时需要返回 true，否则选项菜单将不再显示。

在创建选项菜单 Options Menu 时，首先在 Activity 中重写 onCreateOptionsMenu(Menu menu)方法，然后调用 menu 的 add()方法添加菜单项，add()方法有如下 4 个重载。

(1) add(int groupId, int itemId, int order, CharSequence title)。

（2） add(int groupId，int itemId，int order，int titleRes)。

（3） add(CharSequence title)。

（4） add(int titleRes)。

上述方法都是向 Menu 中添加一个菜单项并返回 MenuItem 对象。其中的 groupId 参数是菜单项所在组的 ID；itemId 参数是唯一标识菜单项的 ID；order 参数代表菜单项显示顺序的编号，编号小的显示在前面；title 参数是菜单项的标题，titleRes 参数是 String 对象的资源标识符。

一般地，MenuItem 对象代表一个菜单项，调用下列方法可以设置菜单项的属性。

（1） setAlphabeticShortcut(char alphaChar)：设置 MenuItem 的字母快捷键。

（2） setNumbericShortcut(char numbericChar)：设置 MenuItem 的数字快捷键。

（3） setIcon(Drawable icon)：设置 MenuItem 的图标。

（4） setIntent(Intent intent)：为 MenuItem 绑定 Intent 对象，当被选中时将会调用 startActivity 方法处理相应的 Intent。

（5） setShortcut(char numbericChar，char alphaChar)：为 MenuItem 设置数字快捷键和字母快捷键，当按下快捷键或按住 Alt 键的同时按下快捷键将会触发 MenuItem 的选中事件。

（6） setTitle(int title 或 CharSequence title)：为 MenuItem 设置标题。

（7） setTitleCondensed(CharSequence title)：设置 MenuItem 的缩略标题，当 MenuItem 不能显示全部的标题时，显示缩略标题。

2. 响应菜单项的单击

Activity 中的 public boolean onOptionsItemSelected(MenuItem item)方法处理菜单项的单击事件，当菜单中某个选项被选中时调用该方法，默认返回 false。

在响应菜单时需要通过 ID 号来判断哪个菜单项被单击了。因此常规的做法是定义一些 ID 常量，但在 Android 中有更好的方法，就是通过资源文件来引用。

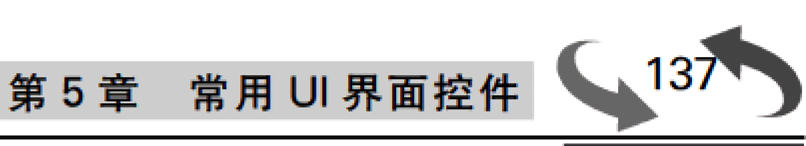
除了用回调方法 onOptionsItemSelected()来处理用户选中菜单事件外，还可以为每个菜单项(即 MenuItem)对象添加 onMenuItemClickListener()方法来监听并处理菜单选中事件。

3. 关闭和移除菜单项

Menu 类提供了一些方法，用来关闭或移除菜单项。调用 close()方法，如果菜单正在显示，则关闭菜单；调用 clear()方法，可以移除菜单中所有子项；调用 removeGroup(int groupId)方法，如果指定 ID 的组不为空，则从菜单中移除该组。调用 removeItem(int id)方法，则移除指定 ID 的 MenuItem。

【例 5-12】 示例工程 05_OptionsMenuExample 演示了 OptionsMenu 的用法。工程中调用不同的 add()方法定义了 3 个选项菜单，并处理了每个菜单项的单击事件。

```
//package 和 import 语句略
public class MainActivity extends Activity {
```




```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //分别用 3种方法添加菜单项
        menu.add("菜单项 1");                          //直接指定标题,ID号=0
        menu.add(1, 1, 3,R.string.MenuItem1);          //通过资源指定标题
        menu.add(1, 2, 3, "菜单项 3");                 //指定菜单项的组号、ID、排序号、标题
        return true;                                   //如果希望显示菜单,返回 true
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
                    //处理用户选中菜单事件
        switch(item.getItemId()) {
            case 0:
                Toast.makeText(MainActivity.this, "您单击了菜单项 1", Toast.LENGTH_LONG).show
                ();
                return true;
            case 1:
                Toast.makeText(MainActivity.this, "您单击了菜单项 2", Toast.LENGTH_LONG).show
                ();
                return true;
            case 2:
                Toast.makeText(MainActivity.this, "您单击了菜单项 3", Toast.LENGTH_LONG).show
                ();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

运行结果如图 5-13 所示。该工程运行后,单击模拟器右侧的 Menu 键或手机上的 Menu 键,会在 Activity 底部弹出设置的菜单项,单击相应的菜单项,会实现相应的功能。

说明:旧版本的 Android 选项菜单在屏幕底部最多只能显示 6 个菜单项,超过 6 个时,第 6 个菜单项会被系统替换为一个称为“更多”的子菜单,显示不下的菜单项都作为“更多”菜单的子菜单项,如图 5-14 所示。在 Android 新版本中菜单采用列表的方式显示,当菜单项太多时,会出现滚动条,滑动滚动条可以选择每个菜单项。

图 5-13 单击 Menu 键后显示的选项菜单

图 5-14 Android 旧版本的选项菜单

5.4.2 子菜单 SubMenu

子菜单提供了一种自然的组织菜单项的方式。在 Android 中,可以通过调用 Menu 类的 addSubMenu(int groupId, int itemId, int order, int titleRes)方法创建子菜单 SubMenu。响应子菜单项的单击同样使用 OnOptionsMenuSelected()方法。

子菜单 SubMenu 继承自 Menu,每个 SubMenu 实例代表一个子菜单。设计子菜单时,通过调用 subMenu 的 add()方法添加子菜单项,根据需要重写 onOptionsItemSelected()方法来响应对应的单击事件。选项菜单和上下文菜单都可以加入子菜单,但子菜单不能嵌套子菜单,Android 中菜单只有两层。

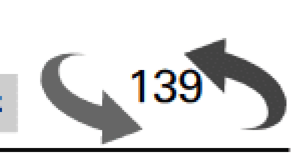
【例 5-13】 示例工程 05_SubMenuExample 演示了如何针对子菜单添加菜单项等内容。主要代码如下:

```
public boolean onCreateOptionsMenu(Menu menu) {
    SubMenu submenu1=menu.addSubMenu(1, 0, 0,"菜单项 1");
                                                //创建子菜单
    SubMenu submenu2=menu.addSubMenu(2, 1, 0,"菜单项 2");
    SubMenu submenu3=menu.addSubMenu(3, 2, 0,"菜单项 3");
    submenu1.add(1, 3, 0, "子菜单项 1");                //增添子菜单项
    submenu1.add(1, 4, 1, "子菜单项 2");
    submenu2.add(2, 5, 0, "子菜单项 3");
    submenu2.add(2, 6, 1, "子菜单项 4");
    submenu3.add(3, 7, 0, "子菜单项 5");
    submenu3.add(4, 8, 1, "子菜单项 6");
    return true;
}
```

代码中通过重写 onCreateOptionsMenu()方法增加菜单,调用 Menu 对象的 addSubMenu(int groupId, int itemId, int order, int titleRes)方法创建子菜单,通过调用 subMenu 对象的 add()方法添加子菜单项。

与选项菜单类似,通过重写 onOptionsItemSelected()方法捕获用户的选择,进而处理用户对菜单项的单击动作。

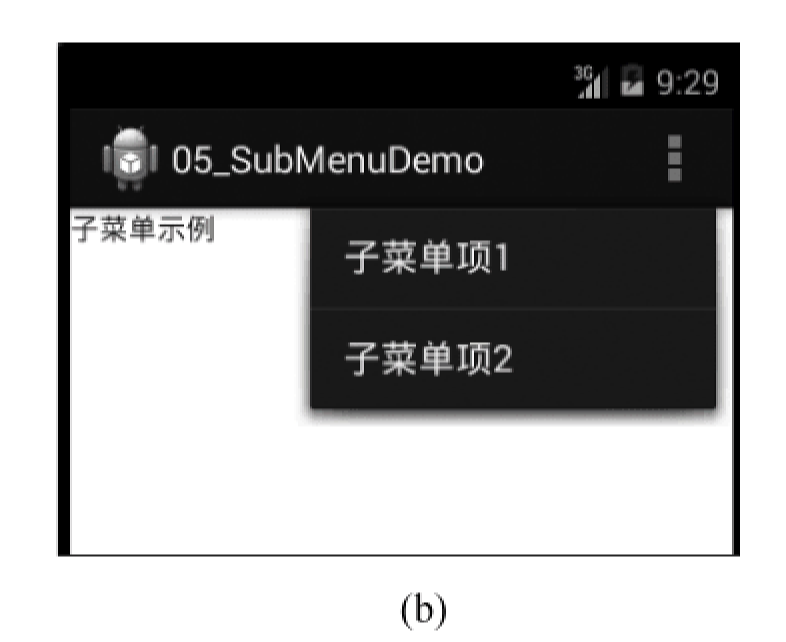
该工程运行后,按下 Menu 键,会在 Activity 底部弹出设置的菜单项。或单击



Activity 窗口右上方的菜单按钮，弹出菜单项，如图 5-15(a)所示，单击“菜单项 1”后弹出子菜单，如图 5-15(b)所示。



(a)



(b)

图 5-15　弹出菜单项

5.4.3　上下文菜单 Context Menu

上下文菜单类似于普通桌面程序中的右键菜单，但在 Android 中不是通过用户右击而得到的，而是当用户长按界面元素超过 2s 后自动出现的菜单。它可以被注册到任何视图对象中，如 ListView 的 item 对象。

创建一个上下文菜单，一般需要重写 Activity 的上下文菜单回调方法 onCreateContextMenu()和 onContextItemSelected()响应菜单单击事件。

onCreateContextMenu()方法的定义如下：

```
onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
```

其中，参数 menu 是创建的上下文菜单；v 是上下文菜单依附的 View 对象；menuInfo 是上下文菜单需要额外显示的信息。

onCreateContextMenu()方法主要用来添加快捷菜单所显示的标题、图标和菜单子项等内容。和选项菜单中的 onCreateOptionsMenu()方法仅在选项菜单第一次启动时被调用一次不同，每次为 View 对象调出上下文菜单时都需要调用该方法。

在 onCreateContextMenu()方法里可以通过调用 add()方法添加相应的菜单项，然后通过 registerForContextMenu(View view)方法为某个 View 对象注册一个上下文菜单 ContextMenu。registerForContentMenu()方法一般在 Activity 的 onCreate()方法里面调用，该方法执行后，会自动为指定的 View 对象添加一个 View. OnCreateContextMenuListener 监听器，这样当长按这个 View 对象时就会弹出上下文菜单。只有通过 registerForContext-Menu()，才能使对应的 View 对象可用上下文菜单。

当用户选择了上下文菜单选项后调用 onContentItemSelected(MenuItem item)方法进行处理，参数中的 item 是被选中的上下文菜单选项。

【例 5-14】 示例工程 05_ContextMenuExample 演示了上下文菜单的设计方法。

Activity 中有两个 EditText 控件，为对象 eTxt1 注册了上下文菜单，主要代码如下：

```
//package 和 import 语句略
```

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.registerForContextMenu(findViewById(R.id.eTxt1));
                            //为 View 对象注册上下文菜单
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,ContextMenu.ContextMenuInfo menuInfo) {
        menu.setHeaderIcon(R.drawable.image1);
        if(v==findViewById(R.id.eTxt1)) {
            menu.add(0, 1, 0, "菜单项 1");
            menu.add(0, 2, 0, "菜单项 2");
            menu.add(0, 3, 0, "菜单项 3");
        }
    }
    @Override                                    //菜单项选中状态变化后的回调方法
    public boolean onContextItemSelected(MenuItem mi) {
        EditText et1= (EditText)this.findViewById(R.id.eTxt1);
        switch(mi.getItemId()) {
            case 1:
                et1.setText(" 您选择了菜单项 1");
                break;
            case 2:
                et1.setText(" 您选择了菜单项 2");
                break;
            case 3:
                et1.setText(" 您选择了菜单项 3");
                break;
          }
        return true;
    }
}
```

该工程运行后，在第 1 个 EditText 控件上长按超过 2s 后自动出现上下文菜单，如图 5-16 所示。单击某一菜单项，会实现相应的处理。而长按第 2 个 EditText 控件不会出现上下文菜单。

5.5 本章小结

本章学习了 Toast、Notification、AutoCompleteTextView 和对话框等提示信息的控件，以及 ListView、Spinner 和 TabHost 等容器类组件的设计方法。还学习了日期和时间



图 5-16　长按后出现的上下文菜单

相关的控件、选项菜单、上下文菜单、子菜单等常用菜单的编程实现方法。学习本章内容时，要求重点掌握各种控件使用方法和相关事件的处理方法。

习　　题

1．设计一个用户登录的对话框，在对话框中输入用户名和密码，单击"确定"按钮关闭对话框。

2．设计一个以 Spinner 方式显示省市列表的应用程序，当用户在第一个下拉列表中选择某个省后，另一个下拉列表中列出对应该省的城市名称供用户选择。

3．设计一个程序，实现动态下拉列表。当用户在 EditText 中写入文本，单击"添加"按钮，能够将其存储在 Spinner 项目中；如果在 EditText 中写入文本，单击"删除"按钮，能够将指定内容的项从 Spinner 项中删除。

4．修改第 3 题的程序，当用户在 EditText 中写入文本，单击"添加"按钮，能够将其存储在 Spinner 项目中；当用户在列表中单击某项时，选项文本显示在 EditText 中，单击"删除"按钮，能够将该项从 Spinner 项中删除。

5．设计一个选择时间的应用程序，要求 Activity 中有一个文本输入框，当用户单击输入框时，弹出"时间选择"对话框，时间对话框中的默认时间为系统当前时间，把用户选择的时间显示在文本输入框中。

6．设计一个用于注册的 Activity。要求界面中的注册项包括用户名、账号、密码、性别、出生年月日、爱好和"注册"按钮；用户名框中只能输入大写字母，宽度为 200 像素；账号框只能输入数字，宽度为 200 像素；密码框不可显示明文，宽度为 150 像素；性别用单选按钮，默认选中"男"；出生年月日使用"日期选择"对话框输入，默认值为当前日期；爱好用多选框实现，至少要有 4 个选项，默认选中第一个和第二个选项；"注册"按钮要水平居中。

7．设计一个应用程序，用户在 EditText 中输入一个 0～100 的数字，单击"转换"按钮后，按照输入数字所属的分数段（90～100，优秀；80～89，良好；70～79，中等；60～69，及格；0～59，不及格），在 TextView 中显示"优秀"、"良好"、"中等"、"及格"或"不及格"，如

果用户输入的数字不符合要求，则弹出“警告”对话框，要求用户重新输入。

8. 设计一个应用程序，用户在 EditText 中输入一个数字，判断该数能否同时被 5 和 7 整除。

9. 当在 Activity 中有多个 View 对象时，如何让其中的某些对象能弹出上下文菜单而其余的没有上下文菜单项？

10. 在 Android 中使用 Menu 时可能需要重写的方法有哪些？这些方法分别在什么情况下被调用？

11. 为第 6 题的 Activity 添加菜单，菜单项为“清空各选项”和“退出”。当用户单击“清空各选项”时，将所有文本输入框的文字清空，所有单选按钮和复选框设为启动时的默认选项。当用户单击“退出”时，关闭 Activity。

12. 设计一个应用程序，界面中有一个 TextView，其中显示有一行文字。为 Activity 添加选项菜单(Options Menu)，包括“红”、“绿”、“蓝”3 个菜单项，用户选择一个菜单项，即将 TextView 中的文字设为相应的颜色。

第6章 组件间的通信和广播

Intent 是 Android 系统的消息传递机制，它能在程序运行的过程中连接两个不同的组件。Intent 在 Android 应用程序的开发中起着重要作用，在启动其他 Activity 或者 Service、发送广播消息、传递数据、调用外部程序时都需要用到。Activity、Service、BroadcastReceiver 等组件之间的交互和通信大都是使用 Intent 完成的。本章主要介绍 Intent 的概念及其在组件通信中的应用，如何利用 Intent 实现 Activity 之间跳转与通信，以及如何利用 Broadcast Intent 发送广播消息和利用 Broadcast Receiver 来接收广播消息。还介绍了 AppWidget 的相关概念和设计方法，AppWidget 利用广播机制接收消息。

6.1 Intent

6.1.1 Intent 及其用途

Intent 的字面含义，是“想要”或“意图”之意，是一个将要执行的动作的抽象描述。例如，在主 Activity 当中，告诉程序想要前往哪里，要移交主动权到哪一个 Activity，这就是 Intent 对象所处理的任务之一。在 Android 系统中，Intent 提供了一种通用的消息机制，它允许在用户的应用程序与其他的应用程序之间传递 Intent 来执行动作和产生事件。

Intent 是一种运行时绑定(runtime binding)机制，它能在程序运行的过程中连接两个不同的组件，用来协助完成各应用或组件间的交互与通信。Intent 负责对应用中一次操作的动作、动作涉及的数据、附加数据等进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将相应数据传递给调用的组件并完成组件的调用。例如，Activity 希望打开网页浏览器查看某一网页的内容，那么只需要发出 WEB_SEARCH_ACTION 请求给 Android，Android 会根据 Intent 的内容，查询各组件注册时声明的 IntentFilter，找到网页浏览器并启动它来浏览网页。

Intent 的一个主要的用途是启动其他 Activity 或者 Service。启动一个新的 Activity 一般通过调用 Context. startActivity()方法或 Context. startActivityForResult()方法来递 Intent。而当需要启动或绑定一个 Service 组件时，则通过调用 context. startService()方法或 context. bindService()方法来传递 Intent。后者的具体使用方法将在第 7 章介绍。

Intent 的另一种重要用途是发送广播消息。应用程序和 Android 系统都可以使用

Intent 发送广播消息，广播消息的内容可以是与应用程序密切相关的数据信息或消息，也可以是 Android 的系统信息，例如网络连接变化、电池电量变化、接收到短信或系统设置变化等。此时一般通过调用 context. sendBroadcast()、context. sendOrderedBroadcast() 或 context. sendStickyBroadcast()方法传递 Intent。当 Broadcast Intent 被广播后，如果应用程序注册了 BroadcastReceiver，其 Intent Filter 过滤条件满足，则可以接收到该广播消息。

总之，Activity 之间可以通过 Intent 对象进行交互，可以通过 Intent 对象启动另外的 Activity、启动 Service、发起广播 Broadcast 等，同时还可以完成数据传递。

Intent 类定义在 android. content 包中，在使用 Intent 对象之前需要引入该包：

```
import android.content.Intent;
```

6.1.2 Intent 对象的属性

一个 Intent 对象由目标组件名称描述 Component、执行动作描述 Action、该动作相关联数据的描述 Data、动作分类描述 Category、数据类型描述 Type、附加信息描述 Extra 及标志 Flag 等几部分组成。它们都是 Intent 对象的属性，这些属性可以在 XML 文件中定义，也可以在 Java 程序中通过 Intent 类的方法来获取和设置。下面分别介绍这些属性及其使用方法。

1. Component

Component(组件名称)属性用于指定 Intent 的目标组件，一般由相应组件的包名与类名组合而成。通常 Android 会根据 Intent 中包含的其他属性的信息，例如 Action、Data、Type、Category 及 Intent filter 的过滤条件进行查找，最终找到一个与之匹配的目标组件。但是，如果 Component 这个属性有指定值的话，将直接使用它指定的组件，而不再执行上述查找过程。

指定了 Component 属性值之后，Intent 的其他属性值都是可选的，此时该 Intent 就是一个显式 Intent；如果不指定 Component 属性值，则该 Intent 就是个隐式 Intent。

2. Action

Action(动作)属性用来指明要实施的动作是什么，其属性值是 Intent 即将触发动作名称的字符串。可使用 SDK 中预定义的一些标准的动作，这些动作由 Intent 类中预先定义好的常量字符串描述。

下面是几个常用的预定义 Action 属性值。

(1) ACTION_MAIN：它在几乎每个 AndroidManifest. xml 配置文件中都会出现，定义在<activity>节点中的<intent-filter></intent-filter>标签之间。它用来标记某个 Activity 为程序的入口，如 android. intent. action. MAIN。该 Action 并不会接收任何数据，同时结束后也不会返回任何数据。

(2) ACTION_CALL：拨打指定的电话号码，电话号码在 Data 属性中用 Uri 格式

表示。

(3) ACTION_VIEW：通常和特定的 Data 和 Uri 相配合使用，用于将数据和网站等信息显示给用户。Android 会根据 Uri 对象提供的数据类型打开相应的 Activity。例如，http:×××会打开浏览器，而 tel:×××会打开拨号程序。

(4) ACTION_DIAL：用于描述给用户打电话的动作，通过和 Data 配合使用将会触发给特定 Data 的用户打电话。

(5) ACTION_EDIT：请求一个 Activity，并可以编辑该 Intent 中 Data 属性的数据。

用户也可以根据需要自行定义 Action 的值，如：edu.hebust.zxm.intent.ACTION_EDIT，自定义的 Action 最好能表明其意义，以方便使用。

调用 Intent 对象的 getAction()方法，可以获取动作字符串；调用 setAction()方法，可以设置动作。

3. Data

Data(数据)属性一般用一个 Uri 变量来表示。Data 主要完成对 Intent 消息中数据的封装，描述 Intent 动作所操作数据的 URI 及 MIME(类型)，不同类型的 Action 会有不同的 Data 封装，如打电话的 Intent 会封装成 tel://格式的 URI，而 ACTION_VIEW 的 Intent 中的 Data 则会封装成 http://格式的 URI。正确的 Data 封装对 Intent 请求的匹配很重要，Android 系统会根据 Data 的 URI 和 MIME 找到能处理该 Intent 的最佳目标组件。

4. Category

这个属性用于描述目标组件的类别信息，是一个字符串对象。它用于指定将要执行的这个 Action 的其他一些额外的信息。例如，LAUNCHER_CATEGORY 表示 Intent 的接收者应该在 Launcher 中作为顶级应用出现，而 ALTERNATIVE_CATEGORY 表示当前的 Intent 是一系列的可选动作中的一个，这些动作可以在同一块数据上执行。

一个 Intent 中可以包含多个 Category(类别)。Android 系统同样定义了一组静态字符常量来表示 Intent 的不同类别。如果没有设置 Category 属性值，Intent 会与在 Intent filter 中包含 android.category.DEFAULT 的 Activity 匹配。

以下是一些常见的 Category 常量。

(1) CATEGORY_BROWSABLE：目标 Activity 能通过在网页浏览器中单击链接而激活。

(2) CATEGORY_GADGET：表示目标 Activity 是可以嵌入到其他 Activity 中的。

(3) CATEGORY_HOME：表示目标 Activity 为 HOME Activity，即手机开机启动后显示的 Activity，或按下 HOME 键后显示的 Activity。

(4) CATEGORY_LAUNCHER：表明目标 Activity 是应用程序中最先被执行的 Activity。

(5) CATEGORY_PREFERENCE：表明目标 Activity 是一个有优先权设置的 Activity。

调用 Intent 对象的 addCategory()方法可以添加一个 Category，调用 removeCategory()

方法可以删除一个已经添加到 Intent 的 Category，调用 getCategories()方法可以得到一个 Category。

5. Type

Type(数据类型)用于显式指定 Intent 的数据类型。一般 Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性，可以强制采用显式指定的类型而不再进行隐式的判定。

6. Extra

Extra(附加信息)是其他所有附加信息的集合。使用 Extra 可以为组件提供扩展信息，例如，如果要执行"发送电子邮件"这个动作，可以将电子邮件的标题、正文等保存在 Extra 里，传给电子邮件发送组件。Extra 属性值以"键-值"对形式保存。

Intent 通过调用 putExtra()或 putExtras()方法来添加一个新的键-值对，而在目标 Activity 中调用 getXxxExtra()或 getExtras()方法来获取 Extra 属性中的键-值对。在 Android 系统的 Intent 类中，同样对一些常用的 Extra 键进行了定义，例如：

(1) EXTRA_BCC：装有邮件密送地址的字符串数组。

(2) EXTRA_EMAIL：装有邮件发送地址的字符串数组。

(3) EXTRA_UID：使用 ACTION_UID_REMOVED 动作时，描述删除用户的 ID。

(4) EXTRA_TEXT：当使用 ACTION_SEND 动作时，描述要发送文本的信息。

利用 Intent 对象的 Extra 属性，可以在组件之间传递一些参数或数据，具体用法见后续介绍。

7. Flag

Flag 属性是一些有关系统如何启动组件的标志位，Android 同样对其进行了封装。

从上述这些属性值及其作用可以看出，Intent 就是一个动作的完整描述，包含了动作的产生组件、接收组件、动作的特征和传递的消息数据。当一个 Intent 到达目标组件后，目标组件会执行相关的动作。

6.1.3 Intent 的解析

Intent 有两种基本用法，即显式 Intent(Explicit Intent)和隐式 Intent(Implicit Intent)。明确指出了目标组件名称的 Intent，称为"显式 Intent"；没有明确指出目标组件名称的 Intent，则称为"隐式 Intent"。显式 Intent 在构造 Intent 对象时就指定接收者；而隐式 Intent 的发送者在构造 Intent 对象时，并不知道也不关心接收者是谁。

显式 Intent 直接指明要启动的组件，即它指定了 Component 属性。一般是通过调用 setClass(Context, Class)方法或 setComponent(ComponentName)方法来指定具体的目标组件类，通知启动对应的组件(如 Service 或 Activity)，此时不需 Android 解析，因为目标已很明确。

例如，以下示例代码片段实现了 MainActivity 到 NextActivity 的跳转。

```
Intent intent=new Intent();
intent.setClass(MainActivity.this,NextActivity.class);
startActivity(intent);                    //启动另一个名为 NextActivity的 Activity
```

在显式 Intent 中，决定目标组件的唯一要素就是组件名称，因此，如果 Intent 中已经明确定义了目标组件的名称，那么就完全不用再定义其他 Intent 内容。

显式 Intent 直接用组件的名称定义目标组件，这种方式很直接。但是由于开发人员往往并不清楚别的应用程序的组件名称，因此，显式 Intent 更多用于在应用程序内部传递消息。如在某应用程序内，一个 Activity 启动另一个 Activity。而隐式 Intent 不使用组件名称定义需要激活的目标组件，它更广泛地用于在不同应用程序之间传递消息。

由于隐式 Intent 没有明确的目标组件名称，所以必须由 Android 系统帮助应用程序寻找与 Intent 请求意图最匹配的组件。具体的选择方法是：Android 将 Intent 的请求内容和一个称为 Intent Filter 的过滤器比较，Intent Filter 中包含系统中所有可能的待选组件。如果 Intent Filter 中某一组件匹配隐式 Intent 请求的内容，那么 Android 就选择该组件作为该隐式 Intent 的目标组件。这个过程称为“解析”。隐式 Intent 需要 Android 进行解析，并将此 Intent 映射给可以处理此 Intent 的 Activity、Service 或 BroadcastReceiver。

一个应用程序组件开发完成后，需要告诉 Android 系统自己能够处理哪些隐式 Intent 请求。利用 Intent Filter 声明该应用程序接受什么样的 Intent 请求就可以实现这一目的。这些声明通常在 AndroidManifest. xml 配置文件中用<intent-filter>元素描述，例如：

```
<intent-filter>
    <action android:name="android.intent.action.WEB_SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

每个<intent-filter>元素描述该组件所能响应 Intent 请求的能力，包括组件希望接收什么类型的请求行为，什么类型的请求数据。例如，请求网页浏览器，网页浏览器程序的 Intent Filter 就应该声明它所希望接收的 Intent Action 是 WEB_SEARCH_ACTION，以及与之相关的请求数据是网页地址。

Intent 解析机制主要是通过查找在 AndroidManifest. xml 配置文件中已注册的所有<intent-filter>及其中定义的 Intent，最终找到匹配的 Intent。在这个解析过程中，必须要进行“动作”、“数据”以及“类别”3 个方面的检查。如果任何一方面不匹配，Android 都不会将该隐式 Intent 传递给目标组件。这 3 方面检查的具体规则如下。

1. Action

Action 主要的内容有 MAIN(程序的进入点)、VIEW、PICK 和 EDIT 等。一般地，一个 Intent 只能设置一种 Action，但是一个 Intent Filter 却可以设置多个 Action。当 Intent Filter 设置了多个 Action 时，只需一个满足，即可完成 Action 验证；当 Intent Filter 中没有说明任何一个 Action 时，任何的 Action 都不会与之匹配。而如果 Intent 中没有包含任何 Action 时，只要 Intent Filter 中含有 Action，便会匹配成功。

2. Data

Data 是用 Uri 的形式来表示的。例如，想要查看一个人的数据时，需要建立一个 Intent，它包含了 VIEW 动作(Action)及指向该联系人数据的 Uri 描述句。对数据的检查主要包含两部分：数据的 Uri 及数据类型，而数据 Uri 又分为三部分，分别是 scheme、authority 和 path，只有这些全部匹配时，Data 的验证才会成功。

(1) 如果 Intent 没有提供 Type，系统将从 Data 中得到数据类型。和 Action 一样，目标组件的数据类型列表中必须包含 Intent 的数据类型，否则不能匹配。

(2) 如果 Intent 中的数据不是“content:”类型的 URI，而且 Intent 也没有明确指定它的 Type，则将根据 Intent 中数据的 scheme(如“http:”或者“mailto:”)进行匹配。同上，Intent 的 scheme 必须出现在目标组件的 scheme 列表中。

3. Category

Intent Filter 同样可以设置多个 Category。当 Intent 中的 Category 与 Intent Filter 中的一个 Category 完全匹配时，便会通过 Category 的检查，而其他的 Category 并不受影响。但是当 Intent Filter 没有设置 Category 时，只能与没有设置 Category 的 Intent 相匹配。

如果 Intent 指定了一个或多个 Category，这些类别必须全部出现在组件的类别列表中。例如，Intent 中包含了两个类别：LAUNCHER_CATEGORY 和 ALTERNATIVE_CATEGORY，则解析得到的目标组件也必须至少包含这两个类别。

6.2 利用 Intent 启动另一个 Activity

如前所述，Intent 寻找目标组件的方法有两种：一种是显式 Intent 通过组件名称直接指定，另一种是隐式 Intent 通过 Intent Filter 过滤指定。

6.2.1 利用显式 Intent 启动另一个 Activity

1. 启动同一个工程中的另一个 Activity

启动工程中的另一个 Activity 是 Intent 很常用的用途。通过调用 context.startActivity()方法或 context.startActivityForResult()方法都可以传递 Intent，启动一个新的 Activity。两者的区别是 startActivityForResult()方法可以接收目标 Activity 返回的参数。

【例 6-1】 工程 06_IntentSameProExample 演示了如何启动同一个工程中的另一个 Activity。

在 MainActivity 中设置了按钮，单击按钮则启动 SecondActivity，即在按钮单击事件的处理代码中启动另一个 Activity。

MainActivity 类的代码如下：




```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        btnstart.setOnClickListener(new OnClickListener() {
                                    //处理按钮的单击事件
          public void onClick(View v) {
            Intent myintent=new Intent(MainActivity.this,SecondActivity.class);
                              //第一个参数是源 Activity,第二个是目标 Activity
            startActivity(myintent);    //启动目标 Activity
            }
        });
    }
}
```

代码被执行后，SecondActivity 将被创建并移到整个 Activity 堆栈的顶部。

需要特别注意的是，为了让应用程序能运行这个新建的 SecondActivity，必须在 AndroidManifest.xml 文件中注册 SecondActivity。具体方法是在<application>元素中添加子元素：

```
<activity android:name=".SecondActivity"/>
```

如果 SecondActivity 与主 Activity 不在同一包中，还需要写明其包路径。

2. 启动不同工程中的 Activity

使用 Intent 还可以启动不同应用程序中的 Activity。

【例 6-2】 在示例工程 06_IntentDiffProExample 中，通过侦听按钮单击事件完成 Activity 跳转，启动示例工程 06_IntentSameProExample 中的 MainActivity。

以下代码被执行后，完成跳转。

```
myintent.setClassName("edu.hebust.zxm.intentsameproexample","edu.hebust.zxm.intentsameproexample.
MainActivity");
startActivity(myintent);                    //启动目标 Activity
```

setClassName()方法的第一个参数是目标应用程序所对应的包名(本例中为 edu.hebust.zxm.intentsameproexample，第二个参数是拟跳转到这个包下的 Activity 名称。与在同一个应用程序中启动 Activity 不同，此时不用修改双方的 AndroidManifest.xml 文件。

6.2.2 利用隐式 Intent 启动另一个 Activity

有时需要将想启动的 Activity 的描述信息放置到 Intent 里面，而不明确指定需要打

开哪个 Activity。例如，一个第三方的 Activity，它只需要描述自己在什么情况下被执行，如果用户启动 Activity 的描述信息正好和这个 Activity 的描述信息相匹配的话，那么这个 Activity 就被启动了。此时一般会用 Uri 对象来描述数据。

一般地，Uri 对象由 scheme、authority 和 path 三部分组成。其 scheme 已经由 Android 所规定，外部调用者可以根据这个标识来判定操作的类别，path 用来指明要操作的具体数据。例如，拨打电话时定义的 Uri 对象为 Uri. parse("tel:13933105035")，其 scheme 为"tel:"；播放音乐时定义的 Uri 对象为 Uri. parse("file:///sdcard/download/everything. mp3")，其 scheme 为"file:"。

【例 6-3】 在示例工程 06_IntentOpenURLExample 中，演示了如何通过 Intent 来打开指定的网址。系统会自动寻找哪一个应用程序适合打开 URL 地址，本例中使用的是内嵌的浏览器 Browser。

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        btnstart.setOnClickListener(new OnClickListener() {
                                            //按钮对应的单击事件
            public void onClick(View v) {
                Uri myuri= Uri.parse("http://www.baidu.com");
                                    //定义 Uri 对象
                Intent myintent= new Intent(Intent.ACTION_VIEW, myuri);
                                    //定义隐式 Intent,第一个参数是动作,第二个参数是数据
                startActivity(myintent);
                                    //启动与 Intent 匹配的 Activity
            }
        });
    }
}
```

示例工程的运行结果如图 6-1 所示。

使用这一方法可以启动 Android 系统提供的很多应用组件。例如，在上述代码中修改 Intent 的相关语句，可以播放 MP3 音频文件。系统会自动寻找适合打开指定音频文件的应用程序，并启动它。

【例 6-4】 示例工程 06_IntentOpenMP3Example 演示了利用 Intent 播放 MP3 音频文件，MainActivity 类的主要代码如下：



图 6-1 打开指定网页

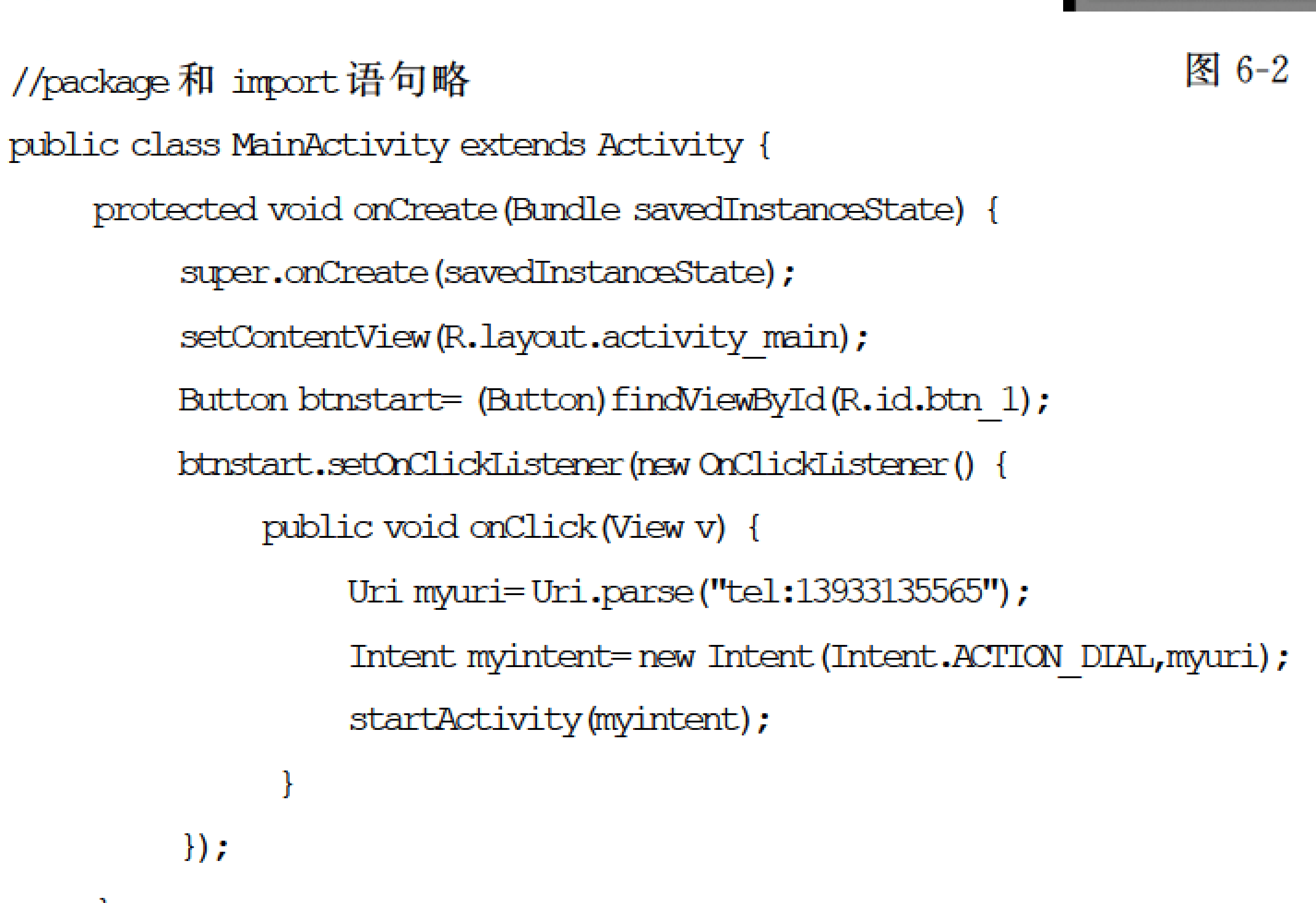
```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        btnstart.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Intent myintent=new Intent(Intent.ACTION_VIEW);
                Uri uri=Uri.parse("file:///storage/sdcard/music/music01.mp3");
                myintent.setDataAndType(uri, "audio/mp3");
                startActivity(myintent);
            }
        });
    }
}
```



图 6-2　播放 MP3 文件

示例工程的运行结果如图 6-2 所示。

【例 6-5】　示例工程 06_IntentTelExample 演示了利用 Intent 实现启动拨打电话界面，并将电话号码 13933135565 设为当前号码。MainActivity 类的主要代码如下：



```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        btnstart.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Uri myuri=Uri.parse("tel:13933135565");
                Intent myintent=new Intent(Intent.ACTION_DIAL,myuri);
                startActivity(myintent);
            }
        });
    }
}
```

示例工程的运行结果如图 6-3 所示。

在上面的这些示例中，编程人员并没有告诉系统需要播放音乐或是拨打电话，只是把对启动 Activity 的描述信息放到 Intent 中，则当执行 startActivity()方法后，系统会自动匹配最适合的应用程序并启动相应的服务。

使用同一方法，还可以实现打开地图、拨打电话、安装或卸载程序、发邮件、发短信、发

彩信等功能,在此不再一一赘述。



图 6-3 启动拨打电话界面

6.3 利用 Intent 在组件之间传递数据

6.3.1 传递单个参数

使用 Intent 类的 putExtra()方法可以将数据参数加入到 Intent 对象中,它采用键-值对(key-value)的形式保存数据。这样,利用 Intent 就可以实现在 Activity 间传递数据。从一个 Activity 跳转到另一个 Activity 时,Intent 中的数据就像参数一样传递给目标 Activity。在目标 Activity 中使用 getXxxExtra()方法取出数据,取出数据时使用"键"找出对应的"值"。

【例 6-6】 示例工程 06_IntentExtrasExample 演示了在 Activity 之间传递单数据。

工程的实现过程如下。

步骤 1:创建 SecondActivity,并在 AndroidManifest.xml 配置文件中声明它。

```
<activity android:name="edu.hebust.zxm.intentextrasexample.SecondActivity"/>
```

步骤 2:在源 Activity 中新建一个 Intent 对象。本例中源 Activity 命名为 MainActivity,Intent 对象命名为 myintent。

```
Intent myintent=new Intent();
```

步骤 3:在 Intent 对象中加入数据,这些数据都以"键-值"对的形式添加到 Intent 对象。本例中,Data 是键,"这是来自 MainActivity 的字符串"是值。

```
myintent.putExtra("Data","这是来自 MainAtivity的字符串");
```

步骤 4：启动目标 Activity，本例目标 Activity 命名为 SecondActivity。

```
myintent.setClass(MainActivity.this, SecondActivity.class);
startActivity(myintent);                                    //启动目标 Activity
```

步骤 5：在目标 Activity 中获取参数值。调用 Intent 对象的 getStringExtra()方法获取键对应的值。

```
String receive=getIntent().getStringExtra("Data");          //得到传过来的数据
```

需要注意的是，本例中传递的数据并不是持久化状态，没有存储在相应的文件中，Activity 退出后，数据就被销毁了。

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    Intent myintent;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnstart=(Button)findViewById(R.id.btn_1);
        myintent=new Intent();
                        //创建 Intent 对象
        myintent.putExtra("Data","这是来自 MainActivity 的字符串");
                        //向 Intent 对象添加数据
        myintent.setClass(MainActivity.this, SecondActivity.class);
        btnstart.setOnClickListener(new OnClickListener() {     //按钮对应的单击事件
            public void onClick(View v) {
                startActivity(myintent);                        //启动目标 Activity
            }
        });
    }
}
```

SecondActivity 类的主要代码如下：

```
//package 和 import 语句略
public class SecondActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        TextView tv_receive=(TextView)findViewById(R.id.tv_bundle);
        String receive=getIntent().getStringExtra("Data");
            //读出 Data 键对应的值
        tv_receive.setText(receive);              //将读出的字符串显示在 TextView 控件中
    }
}
```

示例工程 06_IntentExtrasExample 的运行结果如图 6-4 所示。



(a) (b)

图 6-4　示例工程 06_IntentExtrasExample 的运行结果

6.3.2　传递多个参数

Intent 类的 putExtra()方法不仅可以传递单个键-值对，也可以传递多个键-值对，操作方法与传递单个键-值对类似。

【例 6-7】　工程 06_IntentArgsExample 演示了在 Activity 之间传递两个数据，该程序只需在上进行基础上进行相应的修改即可。

MainActivity 类中的修改：

```
myintent=new Intent();
myintent.putExtra("Data1","这是来自 MainActivity 的第一个字符串");
myintent.putExtra("Data2","这是来自 MainActivity 的第二个字符串");
```

SecondActivity 类中的修改：

```
String receive1=getIntent().getStringExtra("Data1");
String receive2=getIntent().getStringExtra("Data2");
tv_intent.setText(receive1+"\n"+receive2);
```

示例工程 06_IntentArgsExample 的运行结果如图 6-5 所示。



图 6-5　传递多参数示例

6.3.3　利用 Bundle 对象传递参数

Bundle 类在 android.os 包中，其对象常用于携带数据，它也采用键-值对的形式保存数据。虽然其值的类型有一定限制，但常用的 string、int 等数据类型都可以用于 Bundle。

Bundle 类提供了 putXxx()和 getXxx()方法，putXxx()方法用于向 Bundle 对象中放入数据，而 getXxx()方法用于从 Bundle 对象里获取数据。在日常编程中，常用到的方法

主要有 putString()/getString()和 putInt()/getInt()。除此之外，clear()方法用于清除 Bundle 中所有保存的数据，remove()方法移除指定键的数据。

使用 Intent 类的 putExtras()方法可以将 Bundle 对象加入 Intent 对象中。这样，Intent 就可以利用 Bundle 对象实现在 Activity 间传递数据。从一个 Activity 跳转到另一个 Activity 时，Intent 中的 Bundle 对象就像参数一样传递给目标 Activity。

【例 6-8】 工程 06_BundleExample 演示了利用 Bundle 对象在 Activity 之间传递参数。

具体实现步骤如下。

步骤 1：在 AndroidManifest. xml 配置文件中声明 SecondActivity。

```
<activity android:name= "edu.hebust.zxm.buddleexample.SecondActivity"/>
```

步骤 2：在源 Activity 中新建一个 Bundle 对象。本例中源 Activity 命名为 MainActivity，Bundle 对象命名为 myBundle。

```
Bundle myBundle= new Bundle();
```

步骤 3：Bundle 对象中加入数据，这些数据都以键-值对的形式添加到 Bundle 对象。当跳转到目标 Activity 时，就可以从传递过来的 Bundle 对象中取出数据。取出数据时，使用“键”找出对应的“值”。下例中，Data 是键，“这是来自 MainActivity 的字符串”是值。

```
myBundle.putString("Data","这是来自 MainActivity的字符串");
```

步骤 4：新建一个 Intent 对象，并使用 putExtras()方法将该 Bundle 加入这个 Intent 对象。

```
myintent= new Intent();
myintent.setClass(MainActivity.this, SecondActivity.class);
myintent.putExtras(myBundle);
```

步骤 5：目标 Activity 中获取参数值，本例目标 Activity 命名为 SecondActivity。首先调用 getExtras()方法获取 Intent 中的 Bundle 对象，然后调用 Bundle 类的 getString()方法，获取键对应的值。

```
TextView tv_bundle= (TextView)findViewById(R.id.tv_bundle);
Bundle mbundle= getIntent().getExtras();              //得到传过来的 Bundle
String data=mbundle.getString("Data");                //读出 Data 键对应的值
tv_bundle.setText(data);
```

MainActivity 类的主要代码如下：

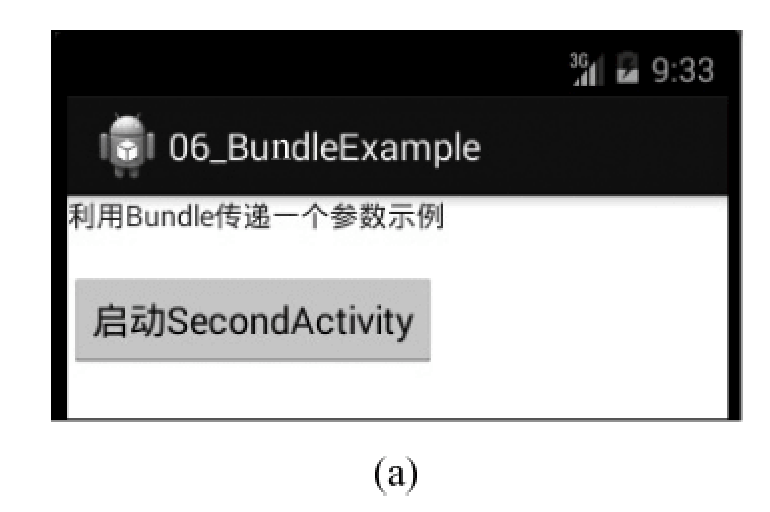
```
//package 和 import 语句略
public class MainActivity extends Activity {
    Intent myintent;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        Button btnstart= (Button) findViewById(R.id.btn_1);
        Bundle myBundle= new Bundle();
        myBundle.putString("Data","这是来自 MainActivity的字符串");
        myintent= new Intent();
        myintent.setClass(MainActivity.this, SecondActivity.class);
        myintent.putExtras(myBundle);
        btnstart.setOnClickListener(new OnClickListener() {        //按钮对应的单击事件
            public void onClick(View v) {
                startActivity(myintent);                        //启动目标 Activity
            }
        });
    }
}
```

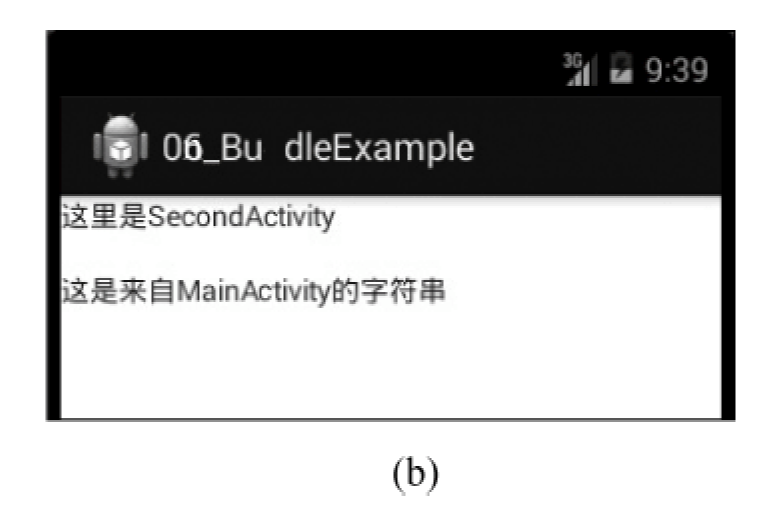
SecondActivity 类的主要代码如下：

```
//package 和 import 语句略
public class SecondActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        TextView tv_bundle= (TextView) findViewById(R.id.tv_bundle);
        Bundle mbundle= getIntent().getExtras();                //得到传过来的 Bundle
        String receive= mbundle.getString("Data");             //读出 Data 键对应的值
        tv_bundle.setText(receive);
    }
}
```

示例工程 06_BundleExample 的运行结果如图 6-6 所示。



(a)



(b)

图 6-6 示例工程 06_BundleExample 的运行结果

Bundle 同样也可以传递多个键-值对。示例工程 06_BundleArgsExample 演示了具体使用方法。在上例的基础上进行相应的修改即可。

MainActivity 类中的修改：

```
Bundle myBundle1= new Bundle();
myBundle1.putString("Data1","这是来自 MainActivity的第 1个字符串");
Bundle myBundle2= new Bundle();
```

```
myBundle2.putString("Data2","这是来自 MainActivity 的第 2 个字符串");
myintent=new Intent();
myintent.setClass(MainActivity.this, SecondActivity.class);
myintent.putExtras(myBundle1);              //将第 1 个键-值对存放在 Intent 实例中
myintent.putExtras(myBundle2);              //将第 2 个键-值对存放在 Intent 实例中
```

SecondActivity 类中的修改：

```
Bundle mbundle=getIntent().getExtras();        //得到 Bundle 中存储的全部信息
String receive1=mbundle.getString("Data1");    //读出数据
String receive2=mbundle.getString("Data2");    //读出数据
tv_bundle.setText(receive1+"\n"+receive2);
```

6.3.4　获取 Activity 的返回值

为了接收目的 Activity 的返回值，源 Activity 在执行跳转的时候不能调用 startActivity()方法，而是要调用 startActivityForResult(Intent, requestCode)方法来启动返回数据的 Activity，该方法的第一个参数是 Intent 对象，包含要到达的 Activity 信息；第二个参数是 requestCode，是唯一标识目标 Activity 的标识码。同一个源 Activity 可能会启动多个目标 Activity，当某一个目标 Activity 返回时，源 Activity 需要判断返回的是哪一个目标 Activity，通过判断参数 requestCode 的值可以实现这一功能。

在目标 Activity 中，调用 setResult()方法设置返回值。该方法有两个参数：resultCode 和表示为 Intent 的结果数据。resultCode 表明运行目标 Activity 的结果状态，其值通常是 Activity.RESULT_OK 或 Activity.RESULT_CANCELED。用户也可以定义自己的 resultCode，它支持任意整数值。当运行目标 Activity 时，如果用户按下硬件返回键，或在调用 finish()方法之前没有调用 setResult()方法，则 resultCode 值将会设定为 Activity.RESULT_CANCELED，结果 Intent 将被设为 null。

当目标 Activity 返回时，会触发调用源 Activity 中的事件处理方法 onActivityResult()，所以通常通过重写源 Activity 中的 onActivityResult()方法来接收目标 Activity 的返回数据。

【例 6-9】 在工程 06_ActivityReturnExample 中，从 MainActivity 跳转到 SecondActivity，SecondActivity 返回时会发送返回数据，返回数据是用户在文本输入框中输入的文字。MainActivity 接收这个返回数据，并显示到自己的 TextView 控件中。

实现步骤如下。

步骤 1：创建并在 AndroidManifest.xml 配置文件中声明 SecondActivity。在 SecondActivity 的 XML 布局文件中添加一个 TextView、一个 EditText 和一个返回按钮。在 SecondActivity 类中实例化控件，获取 TextView 中输入的文字。处理返回按钮的单击事件，返回数据。

SecondActivity 的主要代码如下：

```
//package 和 import 语句略
```

```
public class SecondActivity extends Activity {
    private String backstr;
    private EditText txtreturn;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        txtreturn= (EditText)findViewById(R.id.txt_return);
        Button btnreturn= (Button)findViewById(R.id.btn_return);
        txtreturn.setOnKeyListener(new OnKeyListener() {
            public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
                backstr= txtreturn.getText().toString();
                return false;
            }
        });
        btnreturn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Intent intent= new Intent();
                        //创建 Intent 对象
                intent.putExtra("backstring", backstr);
                        //在 Intent 对象中放入返回值
                setResult(0, intent);
                        //将 Intent 对象传回源 Activity
                finish();                   //结束当前的 Activity,返回
            }
        });
    }
}
```

步骤 2：在 MainActivity 的布局文件中声明一个 Button 和两个 TextView，在 MainActivity 类中实例化控件，处理两个按钮的单击事件。因为要接收 SecondActivity 返回的值，所以跳转的时候调用 startActivityForResult()方法来启动 SecondActivity，并重写 onActivityResult()方法接收返回的数据。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private TextView tvReceive;
    private Intent myintent;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvReceive= (TextView)findViewById(R.id.tv_return);
        Button btnstart= (Button)findViewById(R.id.btn_1);
        myintent= new Intent();
        myintent.setClass(MainActivity.this, SecondActivity.class);
```

```
            btnstart.setOnClickListener(new OnClickListener() {                //按钮对应的单击事件
                public void onClick(View v) {
                    startActivityForResult(myintent, 1);
                    //请求码用于区分目标 Activity,如果只有一个目标 Activity,这个 code 可以为 0
                }
            });
        }
        //重写 onActivityResult()方法接收返回的数据
        protected void onActivityResult(int requestCode,int resultCode,Intent data) {
            super.onActivityResult(requestCode, resultCode, data);
            if(data != null) {
                String receive= data.getStringExtra("backstring");
                                                    //取出返回的数据
                if(requestCode==1) {                //判断是哪一个控件发出的 Intent 请求
                    tvReceive.setText("\n 接收到的返回数据是:"+ receive);
                }
            }
            else{
                tvReceive.setText("没有接收到任何返回数据");
            }
        }
    }
```

注意：在 MainActivity 中的 startActivityForResult()方法有参数 requestCode，SecondActivity 中的 setResult()方法有参数 resultCode，这两个参数的 code 不是对应的。MainActivity 中的 code 用于区分请求的目标 Activity，SecondActivity 中的 code 是用于判断目标 Activity 的返回方式。分别对应 onActivityResult(int requestCode, int resultCode, Intent data)方法中的第一个和第二参数。

示例工程 06_ActivityReturnExample 的运行结果如图 6-7 所示。单击 MainActivity 上的“启动 SecondActivity”按钮，则启动第二个 Activity；单击 SecondActivity 界面中的“返回”按钮，则回到 MainActivity，同时在 TextView 控件上显示 SecondActivity 中 EditText 中的文字。



(a)　　　　(b)

图 6-7　接收 Activity 的返回值

6.4 Broadcast 和 BroadcastReceiver

在 Android 中，广播是一种在应用程序之间传输信息的机制，实现程序间互相通信的功能。如在系统启动、闹钟、来电等情况下，会广播一些消息，其他程序在收到信息后可以进行相应的处理。

一般来说，基于 Broadcast 的应用程序最少要有两个类文件：其中一个是主 Activity，用来发送广播；另一个是广播接收器 BroadcastReceiver，用于收到广播后执行相应动作。BroadcastReceiver 直接继承自 java. long. Object 类，位于 android. content 包。BroadcastReceiver 是对 Broadcast 消息进行过滤、接收并完成响应的组件。

BroadcastReceiver 本身并不实现用户界面，但是当它收到某个广播消息时，可以启动一个 Activity 作为响应。当然，也可以发出 Notification 通知，或启动 Service。BroadcastReceiver 不需要一直运行，当 Android 系统接收到与之匹配的广播消息时，会自动启动相应的 BroadcastReceiver。因此 BroadcastReceiver 也适合做一些资源管理的工作。

6.4.1 发送广播消息

Intent 可以用来在系统范围内广播消息，所以在应用程序组件中，可以构建希望广播的 Intent，然后调用 sendBroadcast() 等方法发送它。发送 Broadcast 和使用 BroadcastReceiver 过滤接收的过程如下。

步骤 1：在需要发送信息的地方，创建一个 Intent，把要发送的信息和用于过滤的信息(如 Action、Category 等)装入这个 Intent 对象。在构造 Intent 时用一个全局唯一的字符串标识其要执行的动作，通常使用应用程序包的名称。

步骤 2：通过调用 Context. sendBroadcast()、sendOrderBroadcast() 或 sendStickyBroadcast()方法，把 Intent 对象以广播方式发送出去。如果要使用 Intent 传递额外数据，可以调用 Intent 的 putExtra()或 putExtras()方法。若调用 sendBroadcast()方法时指定了接收权限，则只有在 AndroidManifest. xml 中用<uses-permission>标签声明了拥有此权限的 BroadcastReceiver 才会有可能接收到发送来的 Broadcast。同样，若在注册 BroadcastReceiver 时指定了可接收的 Broadcast 的权限，则只有在包内的 AndroidManifest. xml 中用<uses-permission>标签声明拥有此权限的 context 对象所发送的 Broadcast 才能被这个 BroadcastReceiver 所接收。

上述提到的 3 个发送方法的不同之处在于：当使用 sendBroadcast() 或 sendStickyBroadcast()方法发送广播时，所有满足条件的接收者会随机地执行；当使用 sendOrderedBroadcast()方法发送广播时，接收者会根据 IntentFilter 中设置的优先级顺序来执行，但相同优先级的 BroadcastReceiver 执行 onReceive()方法的顺序是没有保证的。

步骤 3：当 Intent 发送以后，所有已经注册的 BroadcastReceiver 会检查注册时的 IntentFilter 是否与发送的 Intent 相匹配，若匹配就会调用 BroadcastReceiver 的

onReceive()方法进行接收。所以定义一个 BroadcastReceiver 的时候，一般都需要重写 onReceive()方法。

步骤 4：为应用程序添加适当的权限，并注册 BroadcastReceiver。

步骤 5：等待接收广播并进行相应的处理。在 BroadcastReceiver 接收到与之匹配的广播消息后，onReceive()方法会被调用。

【例 6-10】 工程 06_BroadcastExample 演示了广播消息的发送过程。

首先，要确定发送广播消息的字符串标识 Action。Action 是一个固定格式的字符串，主要是用来区别不同的广播。然后在程序组件中把要广播的消息封装在 Intent 中，并调用广播发送方法 sendBroadcast()、sendOrderedBroadcast()或 sendStickyBroadcast()发送出去。

本例中在 Activity 中定义了一个按钮，在按钮的单击事件处理方法中定义自己的广播信息并将其封装在 Intent 中，用 sendBroadcast()方法发送出去。主要代码如下：

```
private OnClickListener listen_btnStart=new OnClickListener() {
    public void onClick(View v) {
        Intent intent=new Intent("MY_BROADCAST_ACTION");
        //指定广播的 Action 标识,指定了此 Action 的 BroadcastReceiver 会接收此广播
        intent.putExtra("message1", "这是我的广播消息");
         //可选,需要传递参数时可以使用该方法
        intent.putExtra("message2", "这是第二条广播消息");
        sendBroadcast(intent);                //发送广播
    }
};
```

6.4.2　创建并注册 BroadcastReceiver

BroadcastReceiver 用于监听广播消息。创建 BroadcastReceiver 需继承 BroadcastReceiver 类，并重写 onReceive()方法。BroadcastReceiver 类定义在 android. content 包中。重写的 onReceive()方法主要负责广播信息的接收和接收到广播之后的响应操作。

创建 BroadcastReceiver 的一般格式如下：

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        …                                  //接收广播信息,接收到广播之后的响应操作
    }
}
```

为了能够使应用程序中的 BroadcastReceiver 接收指定的广播消息，必须要在 AndroidManifest. xml 文件或在 Java 代码中注册 BroadcastReceiver，并在其中使用 Intent 过滤器指定要处理的广播消息。

在 AndroidManifest. xml 文件中注册 BroadcastReceiver 对象，称为静态注册；在 Java 代码中注册 BroadcastReceiver 对象，称为动态注册。

1. 静态注册

在 AndroidManifest.xml 文件中声明 BroadcastReceiver 可以接收的广播消息。将注册信息定义在<receiver> </receiver>标签中，并通过<intent-filter>标签来设置过滤条件。例如，如下代码声明了 Intent 过滤器的 Action 为 MYBROADTEST，这与发送广播时设置的 Intent 参数相一致，表明这个 BroadcastReceiver 可以接收 Action 为 MYBROADTEST 的广播消息。

```
<receiver
    android:name=".MyBroadcastReceiver"
    android:label="静态广播接收器">
    <intent-filter>
        <action android:name="MYBROADTEST" />
    </intent-filter>
</receiver>
```

【例 6-11】 工程 06_BroadcastReceiverExample 实现了静态注册广播的发送和接收，单击按钮发送广播，通过接收器接收广播，并在 LOG 中显示相关的提示信息。

工程的实现过程如下。

步骤 1：创建 BroadcastReceiver，类名为 MyBroadcastReceiver，重写 onReceive()方法。代码如下：

```
//package 和 import 语句略
public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG="MyBroadcastReceiver";
    public void onReceive(Context context, Intent intent) {
        //重写的 onReceive()方法,主要负责广播信息的接收和接收到广播信息之后的响应操作
        Log.d(TAG, "接收器接收了广播");
        String reseive_action=intent.getAction();                    //获取广播的 Action
        Log.d(TAG, reseive_action);
        String reseive_message=intent.getStringExtra("message");
        Log.d(TAG, reseive_message);
        Toast.makeText(context, "已经接收广播\nAction:"+reseive_action+"\n额外消息:"+reseive
        _message, Toast.LENGTH_LONG).show();
    }
}
```

步骤 2：在 Activity 中创建 MyBroadcastReceiver 类的实例，单击按钮发送广播消息。代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private static final String TAG="MyBroadcastReceiver";
    Button btnS1;
```




```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnS1= (Button)findViewById(R.id.btn_1);
        btnS1.setOnClickListener(listen_btnS1);
        MyBroadcastReceiver myReceiver= new MyBroadcastReceiver();
                                              //实例化 BroadcastReceiver
        Log.d(TAG, "启动静态注册的接收器");
    }
    private OnClickListener listen_btnS1= new OnClickListener() {
        public void onClick(View v) {
            Intent intent= new Intent("MYBROADTEST");   //封装广播消息
            intent.putExtra("message", "这是广播中的额外消息");
                                                    //广播中添加了额外信息
            Log.d(TAG, "发送广播消息");
            sendBroadcast(intent);                  //发送广播
         }
    };
}
```

步骤 3：在 AndroidManifest.xml 文件中注册 BroadcastReceiver。

示例工程的 Activity 界面如图 6-8 所示，单击“发送广播消息”按钮后的运行结果如图 6-9 所示。



图 6-8　Activity 界面



图 6-9　发送和接收广播消息

2. 动态注册

注册 BroadcastReceiver 在 Java 代码中实现。一般需要先创建一个 IntentFilter 对象，并对 IntentFilter 对象设置 Intent 过滤条件，然后在需要注册的地方通过调用 context.registerReceiver()方法来注册监听，通过调用 context.unregisterReceiver()方法来取消监听。此种注册方式的缺点是当 Context 对象被销毁时，该 BroadcastReceiver 也随之被销毁。例如，以下代码注册了 myReceiver 接收器实例，Intent 过滤器的 Action 为 MYBROADTEST，表明这个 BroadcastReceiver 可以接收 Action 为 MYBROADTEST

的广播消息。

```
MyBroadcastReceiver myReceiver=new MyBroadcastReceiver();        //实例化 Receiver
IntentFilter filter=new IntentFilter();                          //实例化 IntentFilter
filter.addAction("MYBROADTEST");                                 //设置过滤器
registerReceiver(myReceiver,filter);                             //注册 Receiver 监听
```

不管是用静态注册还是动态代码注册,在程序退出的时候没有特殊需要都必须要注销它,否则下次启动程序时可能会有多个 BroadcastReceiver。一般在 onStart()方法中注册,执行 registerReceiver()方法;在 onStop()方法中取消注册,执行 unregisterReceiver()方法。

广播接收器的注册和接收都完成之后,一个用户自定义的广播就完成了。

【例 6-12】 工程 06_BroadcastReceiverExample2 实现了动态注册广播的发送和接收,单击按钮发送广播,通过接收器接收广播,并在 LOG 中显示相关的提示信息。

工程的实现过程如下。

步骤 1:创建 BroadcastReceiver,类名为 MyBroadcastReceiver,重写 onReceive()方法。代码与例 6-11 中的 MyBroadcastReceiver 相同。

步骤 2:在 Activity 中创建 MyBroadcastReceiver 类的实例,单击按钮发送广播消息,代码如下:

```
//package 和其他 import 语句略
public class MainActivity extends Activity {
    private static final String TAG="MyBroadcastReceiver";
    Button btnS1;                                                //定义按钮
    MyBroadcastReceiver myReceiver=new MyBroadcastReceiver();
                                                                 //实例化 Receiver
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnS1=(Button)findViewById(R.id.btn_1);
        btnS1.setOnClickListener(listen_btnS1);
    }
    protected void onStart() {
        super.onStart();
          Log.d(TAG, "启动动态注册的接收器");
          IntentFilter filter=new IntentFilter();                //实例化 IntentFilter
          filter.addAction("MYBROADTEST");                       //封装广播消息
          registerReceiver(myReceiver,filter);                   //注册 Receiver 监听
    }
    protected void onStop() {
        super.onStop();
        unregisterReceiver(myReceiver);                          //取消 Receiver 监听
```

```
    }
    private OnClickListener listen_btnS1=new OnClickListener() {
        public void onClick(View v) {
            Intent intent=new Intent("MYBROADTEST");    //封装广播消息
            intent.putExtra("message", "这是广播中的额外消息");
                                                        //广播中添加了额外信息
            Log.d(TAG, "发送广播消息");
            sendBroadcast(intent);                      //发送广播
        }
    };
}
```

从上例中可以看出，静态广播接收器和动态广播接收器的区别在于两者的注册方式不同。静态广播接收器是在 AndroidManifest.xml 配置文件中注册，而动态广播接收器是在 Android 代码中注册。

这里要特别注意：静态广播接收器是常驻型接收器，也就是说当应用程序关闭后，如果有广播消息，接收器就会被系统调用自动运行。而动态广播接收器则不同，它会跟随程序的生命周期结束而结束，当应用程序关闭后，将不会接收到广播消息。

6.4.3 接收系统广播

Broadcast 包括系统广播和自定义广播。系统广播是系统自带的广播事件，不需要用户自己定义就可以直接接收使用，当满足一定条件时，如电池电量低、系统启动完成等，系统会自动发送广播，用户只需要实现广播接收器的注册和接收即可。常见的系统广播事件有系统启动完成、参数设置、时间改变、日期改变、时区改变、电量低、插入或拔出外部媒体、按下媒体按钮、添加包和删除包等，其 Action 值读者可以查阅 API 文档。

系统广播的注册和接收与自定义广播的注册和接收类似，都有广播的注册、发送和接收过程。

【例 6-13】 工程 06_SystemBroadcastExample 演示了如何接收“设置时间”的系统广播消息。设置时间的 Action 值是 android.intent.action.TIME_SET。

首先创建 BroadcastReceiver 对象，接收到系统消息时，弹出 Toast 提示框，代码如下。

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG="我的提示信息";
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "接收器接收了广播");
        String action=intent.getAction();                   //提取接收到广播的 Action 值
        Log.d(TAG, action);
        Toast.makeText(context, "已经接收到系统广播:\nAction="+action, Toast.LENGTH_LONG).show
        ();
```

```
        }
    }
```

在 AndroidManifest.xml 文件中注册 BroadcastReceiver 对象：

```
<receiver
    android:name=".MyBroadcastReceiver"
        <intent-filter>
            <action android:name="android.intent.action.TIME_SET" />
        </intent-filter>
</receiver>
```

程序运行，BroadcastReceiver 对象被注册后，当设置系统时间时，BroadcastReceiver 对象就会接收到系统广播消息，弹出提示框，如图 6-10 所示。



图 6-10 接收到系统广播消息

6.5 主屏幕小部件 AppWidget

AppWidget 是一个具有特定功能的小部件，一般被嵌入到主屏幕（Home Screen）中，用户在不启动任何程序的前提下，就可以在主屏幕上直接浏览 AppWidget 组件所显示的信息或直接运行某个应用程序。本节介绍 AppWidget 的基本概念和设计方法。

6.5.1 AppWidget 简介

AppWidget 是一种可以添加到其他应用程序中的可视化应用程序组件，最典型的用法就是添加到 Android 主屏幕。AppWidget 组件在主屏幕上显示自定义的界面布局，在后台周期性地更新数据信息，并根据这些更新的数据修改主屏幕的显示内容。AppWidget 可以有效地利用手机的屏幕，快捷、方便地浏览信息，为用户带来良好的交互体验。

在 Android 4.0 系统中，自带了多个 AppWidget 组件，如时钟、书签、日历、天气预报、音乐播放器、相框和搜索栏等。在小部件（Widgets）列表中可以查看所有的 AppWidget 组件，如图 6-11 所示。通过长按组件图标，可以将其添加到主屏幕上。例如，长按图 6-11 中的“音乐”图标，将其添加到主屏幕上的结果如图 6-12 所示。





图 6-11　Android 4.0 中的 AppWidget



图 6-12　主屏幕上的 AppWidget

AppWidget 可以使应用程序直接在用户主屏幕上拥有一块交互式的屏幕面板以及一个入口点，它既可以是独立的应用程序，如本机时钟，也可以是更大的应用程序中的一个组件，如 Calendar。

同一个 AppWidget 组件可以根据用户的设置，产生尺寸、布局、刷新速率和更新逻辑等完全不同的副本。这样有助于将 AppWidget 程序设计成多个界面风格的版本，从而适应不同用户的喜好。

AppWidget 通过继承 AppWidgetProvider 类实现控制，而 AppWidgetProvider 类是继承自 BroadcastReceiver 类的，因此每个 AppWidget 就是一个特殊的 BroadcastReceiver。AppWidget 框架通过 Broadcast 和 Intent 与 AppWidget 通信，AppWidget 用 AppWidgetProviderInfo 类来描述细节，其更新使用 RemoteViews 来发送。RemoteViews 被包装成一个 Layout 和特定内容来显示到主屏幕上。

6.5.2　AppWidget 组件的界面布局

AppWidget 是主屏幕上的显示元素，不仅自身具有一定的设计规则，还要与主屏幕上其他的元素保持美观一致。一个典型的 AppWidget 显示在主屏幕上，其结构如图 6-13 所示。最外层是包围盒(Bounding box)边界，这个边界是不同 AppWidget 的分隔界限，这个界限对用户是不可见的。框架(Frame)边界是 AppWidget 背景图像的界限，背景图像会填充满整个框架。最里面是内部图形控件(AppWidget Controls)，这是显示 AppWidget 界面元素的空间。

为了保证多个 AppWidget 显示时不会靠得太近，一般都会设定 AppWidget Margins，这个值是包围盒边界与框架边界的距离。如果 AppWidget Margins 的值为 0，则两个 AppWidget 就会连在一起。在 Android 4.0 中，系统会自动添加 AppWidget

Margins，保证两个 AppWidget 可以保持一定的间隔距离。AppWidget Padding 是框架边界与内部图形控件区域之间的距离，通常设定一个合适的 AppWidget Padding 值，使 AppWidget 的界面元素显示在背景图片的中间区域。

AppWidget 有 6 种标准尺寸，这些尺寸都基于单元格的整数倍。当屏幕纵向时，每个单元格是 80 像素(pixels)宽、100 像素(pixels)高，标准尺寸分别为 4×1、3×3、2×2，当屏幕横向时，每个单元格是 106 像素(pixels)宽、74 像素(pixels)高，标准尺寸分别为：4×1、3×3、2×2。

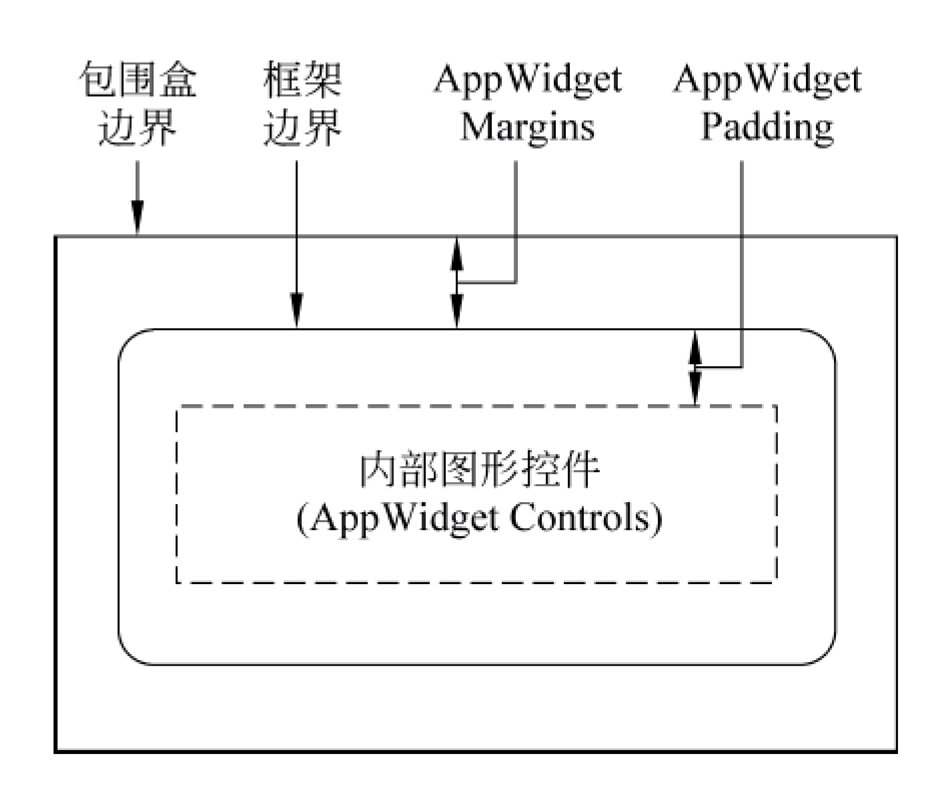


图 6-13　AppWidget 界面结构示意图

AppWidget 与 Activity 的布局设计和实现方法类似，通过在/res/layout 目录中建立基于 XML 的布局资源文件实现。但是由于 AppWidget 布局基于 RemoteViews，同时出于安全和性能考虑，AppWidget 支持的布局和控件存在一些限制。目前 AppWidget 支持的布局有框架布局（FrameLayout）、线性布局（LinearLayout）和相对布局(RelativeLayout)，支持的界面控件有 AnalogClock、Button、Chronometer、ImageButton、ImageView、ProgressBar、TextView、ViewFlipper、ListView、GridView、StatckView 和 AdapterViewFlipper 等。

6.5.3 AppWidget 框架类

AppWidget 框架类包括 AppWidgetProvider、AppWidgetProviderInfo、AppWidgetManager 和 RemoteViews。

1. AppWidgetProvider 类

AppWidgetProvider 类继承自 BroadcastRecevier，负责捕获 AppWidget 广播。AppWidgetProvider 只接收与这个 AppWidget 有关的事件广播，例如该 AppWidget 被更新、删除、启用或禁用。当这些广播事件发生时，AppWidgetProvider 调用相应的方法。其中 onUpdate()和 onReceive()是最常用的方法。

1) onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)方法

在 AppWidgetProviderInfo 中的属性 updatePeriodMillis 定义时间间隔到期时被调

用，主要用来更新 AppWidget 组件的界面显示。除此以外，在用户每次将 AppWidget 拖曳到主屏幕时，该方法也会被调用，一般在此方法中为界面元素定义按钮单击事件处理函数，或者启动一个临时的 Service 进行数据获取等。

Widget 元数据中的 updatePeriodMillis 属性无法进行频繁更新，对于低于 30min 的设定值，该属性并不生效。在 Widget 中如果需要进行频繁更新，一般采用 Service 周期性更新 Widget 的方法。另外，当进行 Widget 更新时，如果在 onUpdate() 函数中代码运行时间超过 5s，例如进行网络操作、复杂运算等，则会产生“应用程序无响应”错误。使用 Service 更新 Widget 可以避免这种问题的出现，将比较耗时的代码在 Service 中实现，然后直接在 Service 中更新 Widget 的界面。

2）onDeleted(Context context，int[] appWidgetIds)方法

当一个 AppWidget 从主屏幕上被删除时调用，一般在此方法中回收资源。

3）onEnabled(Context context)方法

在首个 AppWidget 实例被创建并添加到主屏幕时被调用。AppWidget 可以在主屏幕上创建多个实例，但只有在第一个 AppWidget 实例被创建时才调用该函数。onEnabled()一般用来进行一些初始化工作，例如与数据库建立连接，或者执行对所有 AppWidget 实例来说只需进行一次的设置。

4）onDisabled(Context context)方法

在最后一个 AppWidget 实例被删除时调用，用来释放在 onEnabled()中使用的资源，如删除在 onEnabled()函数中创建的临时数据库。

5）onReceive(Context context，Intent intent)方法

因为 AppWidgetProvider 是继承自 BroadcastReceiver 类，所以需要重写 onReceive()方法。接收广播时该方法被调用。当然必须在 AndroidManifest. xml 文件中将该 AppWidgetProvider 实例注册为一个 Receiver。

将 Widget 添加到主屏幕上，或者从主屏幕删除 Widget 都会引发 AppWidgetProvider 中的事件处理方法。当 AppWidget 第一次添加到主屏幕时，系统会按顺序调用 onEnable()和 onUpdate()，当再次向主屏幕添加 AppWidget 时，系统则仅调用 onUpdate()，当从主屏幕删除 AppWidget 时，如果主屏幕还有这个 AppWidget 实例，则系统仅调用 onDelete()，如果被删除的是这个 AppWidget 的最后一个实例，则系统在调用 onDelete()后会调用 onDisable()。

2. AppWidgetProviderInfo

AppWidgetProviderInfo 描述 AppWidget 的大小、更新频率和初始界面等信息。AppWidgetProviderInfo 信息由一个<appwidget-provider>元素来描述，例如：

```
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="150dp"
  android:minHeight="60dp"
  android:resizeMode="horizontal|vertical"
  android:minResizeHeight="80dp"
```

```
    android:minResizeWidth="48dp"
    android:updatePeriodMillis="3600000"
    android:initialLayout="@layout/appwidget"
    android:previewImage="@drawable/icon_rabit"
/>
```

<appwidget-provider>元素的内容包括 AppWidget 的尺寸、更新周期、布局文件位置、预览图片、拉伸方向和配置界面等。<appwidget-provider>元素通常保存在/res/xml/文件夹中的 XML 文件中，在 AndroidManifest.xml 文件中配置 receiver 参数时在<meta-data>标签中用 resource 属性声明。

1）minWidth 和 minHeight 属性

Android 系统将主屏幕划分为单元格，单元格的大小和数量会随设备的变化而完全不同，一般智能手机会被划分为 4×4 的单元格，而平板电脑会被划分为 8×7 的单元格。当用户将 AppWidget 加入到主屏幕时，AppWidget 会占据一定数量的单元格，占据单元格的数量由 minWidth 和 minHeight 属性值决定。minWidth 和 minHeight 属性值分别表示默认情况下 AppWidget 的显示宽度和高度，也就是 AppWidget 在拖曳到主屏幕时的尺寸。在设定 minWidth 和 minHeight 时，最基本的原则是使 Widget 处于最佳的显示状态。

2）resizeMode 属性

resizeMode 属性用于声明 Widget 的尺寸是否可变，有 4 种方式可供选择，这 4 种方式的参数分别为 none、horizontal、vertical、horizontal|vertical，对应含义分别是不可调整、水平方向调整、垂直方向调整、水平与垂直方向调整。

minResizeWidth 和 minResizeHeight 属性值决定 AppWidget 的最小尺寸。

3）updatePeriodMillis 属性

updatePeriodMillis 属性值表示以毫秒为单位的更新周期，Android 会以这个速率唤醒设备以便更新 AppWidget，开发人员应尽可能地降低设备被唤醒的次数，以降低设备的能量消耗。当更新周期小于 30min 时，Android 系统不会按照此参数更新 Widget，如果需要频繁更新 AppWidget，可以在 Service 服务中实现。

4）initialLayout 属性

initialLayout 属性用来指定 Widget 的布局。

5）previewImage 属性

previewImage 属性定义了在 Android 系统 Widget 列表中预览图像，如果不设置该值，则以程序的图标作为预览图像。

3. AppWidgetManager 和 RemoteView 类

AppWidgetManager 类负责管理和更新 AppWidge，向 AppwidgetProvider 发送通知。

RemoteView 类是一个可以在其他应用进程中运行的类，向 AppWidgetProvider 发送通知。RemoteView 类用作在另一个应用程序进程中托管的 View 层次的代理，从而允

许修改运行在另一个应用程序中的 View 的属性或调用该 View 的方法。例如，一个 AppVidget 使用的 UI 托管在它自己的进程(即主屏幕)中，要从运行在应用程序进程中的 AppWidget Provider 修改这些 View，就需要使用 RemoteView。RemoteView 是构造 AppWidget 的核心。

如果要修改组成 AppWidget UI 的 View 外观，可以创建并修改 RemoteView，然后使用 AppWidgetManger 应用更改。常见的修改操作包括改变 View 的可见性、文本或图像，以及添加 ClickListener。

6.5.4　AppWidget 的设计步骤

AppWidget 的一般设计步骤包括：设计 AppWidget 的布局，设置 AppWidgetProviderInfo 参数，定义 AppWidgetProvider 的子类以实现 AppWidget 的添加、删除、更新、在 AndroidManifest. xml 文件中配置 Receiver。

1. 界面布局和 AppWidgetProviderInfo 参数

AppWidget 的布局设计和实现方法与 Activity 类似，需要在/res/layout 目录中建立基于 XML 的布局资源文件。

为了增加 AppWidget 对不同屏幕尺寸和单元格尺寸的适应性，应该尽量使用具有自适应能力的布局，例如线性布局、相对布局或框架布局。当 AppWidget 的尺寸不够填充满所应占的单元格时，AppWidget 会在横向和纵向拉伸，以填充所有应该占据的单元格。

背景图片通常放置在/res/drawable 目录中，最好使用 PNG 格式图片，这种格式的图片可以自动填满框架(Frame)的空间。

为了适应不同 Android 系统版本对于 widget_margin 参数的不同处理方式，一般添加两个 dimension 资源，第一个在/res/values/dimens. xml 文件中，为较早的 Android 系统提供自定义的 Margins，例如：

```
<dimen name="widget_margin">15dp</dimen>
```

另一个 dimension 资源定义在/res/values-v14/dimens. xml 文件中，为 Android 4. 0 系统设定 widget_margin 参数。因为在 Android 4. 0 中，系统会自动添加 Widget Margins，所以通常将其值设为 0。

除了设计界面布局以外，还需要设置 AppWidgetProviderInfo 参数，这通常在/res/xml/文件夹中的一个 XML 文件中用<appwidget-provider>元素的属性值描述。

在界面设计过程中可以使用下载的 AppWidget 模板包，模板包里面包括 NinePatch 图像文件、XML 文件和 Photoshop 源文件等内容，适用于不同屏幕分辨率和 Android 版本系统，下载地址为 http://developer. android. com/shareables/app_widget_templates-v4. 0. zip。

2. 编写 AppWidgetProvider 类的子类

实现 AppWidget 的添加、删除、更新等过程，主要是通过 AppWidgetProvider 类来实

现。所以设计自己的 AppWidget 必须添加一个 AppWidgetProvider 类的子类。AppWidgetProvider 本身继承自 BroadcastReceiver，用来接收与 Widget 相关的更新、删除、生效和失效等消息，当 AppWidgetProvider 接收这些消息后，会分别调用相应的事件处理方法。

设计该类时，需要重写 onUpdate()、onDeleted()、onEnabled()和 onDisabled()方法。其中 onUpdate()方法在将 AppWidget 拖曳到主屏幕和 updatePeriodMillis 时间间隔到期时被调用，主要用来更新 AppWidget 组件的界面显示、为界面元素定义按钮单击事件处理方法等，是最为重要的一个方法。

3. 在 AndroidManifest.xml 文件中配置 receiver

Android 系统是通过 Broadcast 来通知每一个 AppWidget 的，所以 AppWidget 需要在 AndroidManifest.xml 文件中配置 receiver。例如：

```
<receiver android:name=".MyWidgetProvider">
    <meta-data
       android:name="android.appwidget.provider"
       android:resource="@xml/appwidget_provderinfo" />
    <intent-filter>
       <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
</receiver>
```

其中：

<receiver>标签中的 name 属性定义了 AppWidgetProvider 的子类。

<meta-data>标签中的 name 属性值为 android.appwidget.provider，表示这是一个 AppWidget。resource 属性声明了对<appwidget-provider>元数据 XML 资源的引用，该 XML 文件描述了 AppWidget 的参数设置。

<intent-filter>标签中声明接收 ACTION_APPWIDGET_UPDATE 消息。

4. 安装、加载和删除 AppWidget

安装 AppWidget 与安装其他应用程序相似，通过 Eclipse 上的运行(Run)按钮启动程序的编译、链接、打包和安装过程，唯一区别是在 AppWidget 安装到模拟器后，不会直接出现在主屏幕上，而是加入到小部件(Widget)列表中，需要用户在列表中手动将 AppWidget 添加到主屏幕上。

长按 AppWidget 的预览图标，就可以把 AppWidget 实例加载到主屏幕上，默认情况下占据 3×1 个单元格。在主屏幕上，长按 AppWidget 实例，其边缘出现 4 个控制点，通过拖曳这些控制点，可以调整 AppWidget 的尺寸。

如果希望添加第二个 AppWidget 实例，过程与添加第一个 AppWidget 实例的过程完全一样。

如果希望删除 AppWidget，长按主屏幕上的 AppWidget 实例，主屏幕上方会出现垃



圾桶，直接将 AppWidget 实例拖到垃圾桶即可。需要注意的是主屏幕上的垃圾桶是隐藏的，需要通过长按主屏幕上的 AppWidget 实例才会出现。当拖拽 AppWidget 图标在垃圾桶上方呈现出红色时，松开手指便可完成了删除操作。

【例 6-14】 工程 06_AppWidgetExample 实现了一个 AppWidget，显示当前时间，并每小时更新一次显示时间。

工程的实现过程如下。

步骤 1：设计 AppWidget 的布局文件。

本例中布局文件的文件名为 appwidget.xml，背景图片为/res/drawable/background.png。内部包含 ImageButton 和 TextView 控件，使用线性水平布局。

appwidget.xml 的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:background="@drawable/background"
    android:padding="8dp"  >
    <ImageButton
        android:id="@+id/img_btn"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:src="@drawable/icon_rabit"
        android:layout_gravity="center_vertical"/>
    <TextView  android:id="@+id/showtext"
        android:layout_width="wrap_content"
        android:layout_height="72dp"
        android:textColor="#ffff00"
        android:background="#0000ff"
        android:layout_weight="1"
        android:textSize="30px"
        android:layout_gravity="center_vertical"/>
</LinearLayout>
```

步骤 2：分别在/res/values/dimens.xml 文件和/res/values-v14/dimens.xml 文件中添加两个 dimension 资源，定义 Widget Margins。

步骤 3：创建/res/xml/appwidget_providerinfo.xml 文件，定义 AppWidgetProviderInfo 类信息。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="150dp"
```

```
android:minHeight="60dp"
android:resizeMode="horizontal|vertical"
android:minResizeHeight="80dp"
android:minResizeWidth="48dp"
android:updatePeriodMillis="3600000"
android:initialLayout="@layout/appwidget"
android:previewImage="@drawable/icon_rabit"     />
```

属性 initialLayout 用来指定 Widget 的布局,本例指定为/res/ layout/appwidget.xml 文件。属性 previewImage 定义了在 Android 系统 AppWidget 列表中预览图像,本例指定为/res/ drawable/icon_rabit.jpg 文件,预览效果如图 6-14 所示。



图 6-14　AppWidget 预览图像

步骤 4:编写 AppWidgetProvider 类,实现 AppWidget 的添加、删除和更新。

在本例中,添加继承自 AppWidgetProvider 类的 MyWidgetProvider 类,MyWidgetProvider.java 文件的主要代码如下:

```
//package 和其他 import 语句略
public class MyWidgetProvider extends AppWidgetProvider {
    private static final String TAG="MYWIDGET";
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        final int N=appWidgetIds.length;
        for(int i=0; i<N; i++) {
          int appWidgetId=appWidgetIds[i];
          RemoteViews views=new RemoteViews(context.getPackageName(),R.layout.appwidget);
          java.text.DateFormat df=new java.text.SimpleDateFormat("hh:mm:ss");
          views.setTextViewText(R.id.showtext, "当前时间:\n"+df.format(new
          Date()));
          appWidgetManager.updateAppWidget(appWidgetId, views);
```

```
            }
        }
    }
```

步骤 5：在 AndroidManifest.xml 文件中配置 Receiver。

步骤 6：安装和加载 AppWidget。

经过编译、链接、打包和安装，示例程序加入到小部件(Widgets)列表中，长按图标，就可以将其加载到主屏幕上，如图 6-15 所示。



图 6-15　加载到主屏幕上的示例 AppWidget

6.6　本 章 小 结

本章学习了 Intent 的概念及其在组件通信中的应用。Intent 能在程序运行的过程中连接两个不同的组件，所以广泛用于页面跳转、传递数据、调用外部程序等。使用 Intent 有显式和隐式两种主要方式，前者主要用于不同 Activity 的跳转，后者主要用于 Service、Broadcast 等。本章还学习了如何利用 Intent 启动一个新的 Activity、利用 Intent 在 Activity 之间传递数据、利用 Intent 发送广播消息和使用 BroadcastReceiver 来处理广播事件的过程，以及 AppWidget 的设计方法。设计 AppWidget 一般包括设计布局、定义 AppWidget 参数、定义 AppWidgetProvider 的子类、在 AndroidManifest.xml 文件中配置 Receiver 等过程。学习本章内容时，要求读者重点掌握 Intent 的运行机制和在组件通信中的应用，以及广播消息的发送和接收。

习　题

1. Intent 是什么？在 Android 中其主要用途是什么？

2. 一个 Intent 对象由哪几部分组成？它们的作用分别是什么？

3. 显式 Intent 和隐式 Intent 有什么区别？

4. 简述利用显式 Intent 启动 Activity 的步骤。

5. 简述利用隐式 Intent 启动 Activity 的步骤。

6. 设计一个应用程序，单击初始 Activity 中的按钮后能跳转到另一个 Activity，在目标 Activity 中设定了 TextView 内容以及"欢迎光临"的标题。

7. 设计一个基于多选按钮的 Activity 间跳转的程序，要求初始 Activity 中有一组歌手名字的列表，用户选择后，单击"确定"按钮，则跳转到目标 Activity，目标 Activity 中显示出用户选择的所有歌手名字。

8. 设计一个 Activity 间跳转的程序。要求初始 Activity 中有 3 个目标 Activity 的单选按钮，选择其中一个，单击"确定"按钮，则跳转到指定 Activity。所有目标 Activity 在返回时传递一个标识字符串，返回后在初始 Activity 中显示这个标识字符串。

9. 查阅 API 文档，研究一下 Intent 传递数据时，哪些数据类型可以被传递？

10. 设计一个应用程序，要求用户输入用户名和密码，当用户输入正确的用户名和密码，单击"登录"按钮后，发送广播消息"有用户登录入系统！"；当用户输入用户名和密码有误，单击"登录"按钮后，发送广播消息"有非法用户试图登录入系统，被拒绝！"。

11. 设计一个 BroadcastReceiver 并启动它，接收第 10 题中的广播消息。

12. 简述 BroadcastReceiver 对象的静态注册和动态注册有什么不同，分别用两种注册方法实现第 11 题，体会两者的不同。

13. 简述 AppWidget 的设计原则和注意事项。

14. 运行示例工程 06_AppWidgetExample，分别重写 onDeleted()、onEnabled()和 onDisabled()等方法，在其中添加 LOG 语句，观察 Eclipse 中 LogCat 的输出信息，分析用户对 AppWidget 进行不同操作所引发的事件处理方法，以及其调用顺序关系。

15. 设计一个动态显示照片的 AppWidget，照片更新时间间隔为 1h(小时)。

第7章 Android的后台服务

服务(Service)是Android系统中4个应用程序组件之一,可以在不显示界面的前提下在后台运行指定的任务。本章主要介绍Service的概念及其启动、停止的方法,以及如何获得和使用系统服务。Service和第6章介绍的BroadcastReceiver综合使用可以实现更多、更复杂的功能。

7.1 Service及其生命周期

7.1.1 Service简介

Activity的主要作用是提供UI界面,与用户交互等,而Service是运行在后台的长生命周期的、没有UI界面的Android组件。它相当于在后台运行的Activity,只是不像Activity那样提供与用户交互的界面。当应用程序不需要显示一个与用户交互的界面但是需要其长时间在后台运行时,可以使用Service,如在后台完成数据计算、后台音乐播放等。和Activity不同的是,Service不能自己运行,它一般需要通过某一个Activity或者其他Context对象来调用,如Context.startService()和Context.bindService()等。Service既可以运行在自己的进程中,也可以运行在其他应用程序进程的上下文(context)里面。

Service通常要与Activity联合使用来实现一个完整的应用。例如,在一个媒体播放器程序中,一般由一个或多个Activity来供用户选择歌曲并播放它。但是,因为用户希望退出媒体播放器界面导航到其他界面时,音乐应该继续播放,所以音乐的回放就不能使用Activity了。这种情况下,Activity要调用Context.startService()启动一个服务来在后台运行保持音乐的播放,系统将保持这个音乐回放服务的运行直到它结束或被停止。

Service可分为本地服务(Local Service)和远程服务(Remote Service)两类。本地服务用于应用程序内部,它可以启动并运行,直至有人停止了它或它自己停止。本地服务主要用于实现应用程序自己的一些耗时任务,例如查询升级信息,不占用应用程序所属线程,而是在另一个线程后台执行,这样用户体验会比较好。远程服务用于Android系统内部的应用程序之间,它可以通过自己定义并暴露出来的接口进行程序操作。客户端建立一个到服务对象的连接,并通过那个连接来调用服务。远程服务可被其他应用程序复用,例如,天气预报服务,其他应用程序不需要再写这样的服务,调用已有的即可。

Service 不能自己运行，需要通过调用 Context. startService()或 Context. bindService()方法启动服务。以调用 Context. stopService()或 Context. unbindService()结束。也可以调用 Service. stopSelf()或 Service. stopSelfResult()来使服务自己停止。

Service 一般没有用户操作界面，它运行于系统中不容易被用户发觉，可以使用它开发监控类的程序。

7.1.2 Service 的生命周期

一个 Service 实际上是一个继承自 android. app. Service 的类。Service 与 Activity 一样，也有一个从启动到销毁的过程。当第一次启动 Service 时，先后调用 onCreate()、onStartCommand()这两个方法，当停止 Service 时，则调用 onDestroy()方法。这里需要注意的是，如果 Service 已经启动了，再次启动 Service 时，不会再调用 onCreate()方法，而是直接调用 onStartCommand()方法。

一个 Service 只会创建一次，销毁一次，但可以开始多次，因此，onCreate()和 onDestroy()方法只会被调用一次，而 onStartCommand()方法会被调用多次。

Service 不能自己运行，需要通过调用 Context. startService()或 Context. bindService()方法启动服务。这两个方法都可以启动 Service，它们的使用场合有所不同。

调用 startService()方法启用服务，调用者与服务之间没有关连，即使调用者退出了，服务仍然运行。Service 的生命周期如图 7-1(a)所示，如果服务未被创建，系统会先调用服务的 onCreate()方法，接着调用 onStartCommand()方法，Service 进入运行状态。如果调用 startService()方法前服务已经被创建，多次调用 startService()方法并不会导致多次创建服务，但会导致多次调用 onStartCommand()方法。

调用 startService()方法启动的服务，只能调用 stopService()方法结束服务，服务结束时会调用其 onDestroy()方法。不论调用了多少次 startService()方法，只需要调用一次 stopService()来停止服务。需要注意的是，通过 startService()启动 Service 后，即使调用 startService()的进程结束了，Service 仍然还存在，直到有进程调用 stopService()或者 Service 通过 stopSelf()方法终止时才能结束。如果直接退出而没有调用 stopService()，Service 会一直在后台运行。例如，音乐播放器在后台播放音乐时就处于这种状态。

调用 bindService()方法启用服务，调用者与服务绑定在了一起，调用者一旦退出，服务也就终止。Service 的生命周期如图 7-1(b)所示，onBind()方法只有采用 bindService()方法启动服务时才会调用。该方法在调用者与服务绑定时被调用，当调用者与服务已经绑定，多次调用 bindService()方法并不会导致该方法被多次调用。采用 bindService()方法启动服务时只能调用 onUnbindService()方法解除调用者与服务的绑定，服务结束时会调用 onUnbind()和 onDestroy()方法。

上述两种方式可以混合使用，一个 Service 可以同时启动并且绑定。在这种情况下，如果 Service 已经启动了或者 BIND_AUTO_CREATE 标志被设置，系统会一直保持 Service 的运行状态。如果先调用 startService()启动一个服务，然后再调用 bindService()方法绑定服务，服务仍然会成功绑定到 Activity 上，但在 Activity 关闭后，服务虽然会被解除绑定，但并不会被销毁，也就是说，Service 的 onDestroy()方法不会被调用。例如，音乐播放器后

台工作的 Service 通过 context.startService()启动某个特定音乐播放，但在播放过程中如果用户需要暂停音乐播放，则通过 context.bindService()获取服务连接和 Service 对象，进而通过调用 Service 的对象中的方法来暂停音乐播放并保存相关信息。

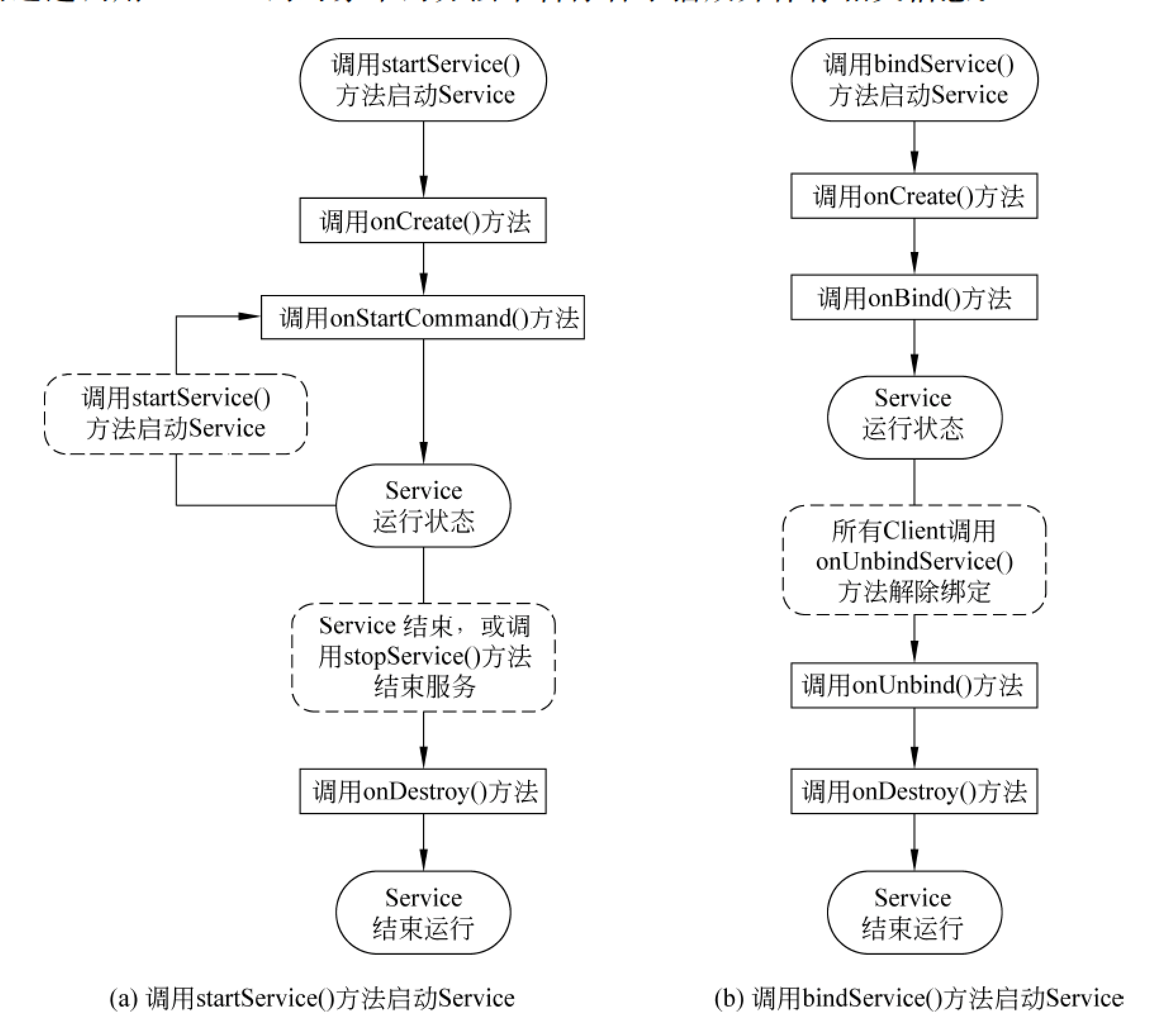


图 7-1　Service 的生命周期

7.2　创建和控制 Service

7.2.1　创建、启动和停止 Service

1. 创建 Service

创建一个 Service 类，必须要继承自 android.app.Service 或它的子类，并重写 onCreate()方法。这个方法中通常进行一些初始化处理。以下示例代码描述了创建一个 Service 类的框架：

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
```

```
public class MyService extends Service {
    @Override
    public void onCreate() {                    //这个方法会在 Service 创建时被调用
        super.onCreate();
        ...                                     //初始化处理
    }
}
```

当创建一个新的 Service 后，必须将这个 Service 在 AndroidManifest.xml 配置文件中注册，方法是在<application></application>节点内包含一个<service>的标签。格式如下：

```
<service
    android:enabled="true"
    android:name=".Service 名字" />
```

如果要确保这个 Service 只能由自己的应用程序启动和停止，则需要在节点下增加一个 permission 属性，例如：

```
<service
    android:enabled="true"
    android:name=".Service 名字"
    android:permission="edu.hebust.zxm.serviceexample.MY_SER_PERMISSION"   />
```

添加了这个 permission 属性后，任何想要访问这个 Service 的第三方应用程序都需要在它的 AndroidManifest.xml 配置文件中包含一个 uses-permission 属性，并且属性值与 Service 中的设置的权限字符串相同。

2. 封装由 Service 执行的任务

Service 执行的任务通过重写 onStartCommand()方法实现。在这个方法中还可以指定 Service 的重新启动行为。当通过调用 startService()方法来启动一个 Service 时，就会调用它的 onStartCommand()方法。如图 7-1(a)所示，这个方法可能在 Service 的生命周期中被执行很多次。

onStartCommand()方法是在 Android 2.0 之后才引入的，替代之前使用的 onStart()方法。onStartCommand()方法提供了和 onStart()方法相同的功能，同时还允许用户告诉系统，如果系统在显式调用 stopService()方法，或 stopSelf()方法之前终止了 Service，那么应该如何重新启动 Service 呢？以下代码片段描述了 onStartCommand()方法的内容。

```
public int onStartCommand(Intent intent, int flags, int startId) {
    startBackgroundTask(intent,startId);
    return Service.START_STICKY;
}
```

onStartCommand()方法的返回值控制了当 Service 被运行时终止后，重新启动

Service 的方式。通过返回以下 Service 常量就可以控制重启模式：

(1) START_STICKY：标准的重新启动方式，与 Android 2.0 之前版本中重写 onStart()方法实现的处理方法相似。当启动 Service 时，将会调用 onStartCommand()方法，但此时传入的 Intent 参数是 null。

(2) START_NOT_STICKY：这种模式适用于处理特殊操作和命令的 Service。通常当操作或命令执行完后，Service 会调用 stopSelf()方法终止自己。当 Service 被运行时终止后，只有当存在未处理的启动调用时，才会重新启动。如果在此之后没有进行 startService()调用，那么该 Service 将停止运行，而不会调用 onStartCommand()方法。对于某些特殊处理要求，例如处理更新、网络轮询等，这种模式非常适用。当停止 Service 后，会在下一个调度间隔中尝试重新启动，而不会在存在资源竞争时重新启动 Service。

(3) START_REDELIVER_INTENT：这种模式是前两种的组合。如果 Service 被运行时终止，那么只有当存在未处理的启动调用，或进程在调用 stopSelf()方法之前被终止时，才会重新启动 Service。后一种情况中，将会调用 onStartCommand()方法，并传回没有正常处理的 Intent。有些 Service 的任务实时性要求较高，必须要确保它请求的命令得以全部执行，适用于这种模式。

需要注意的是，在处理完成后，都要求使用 stopService()方法或 stopSelf()方法显式地停止 Service。

默认情况下，Service 是在应用程序的主线程中启动的，这意味着在 onStartCommand()方法完成的任何处理都是运行在 GUI 主线程中的。实现 Service 的标准模式是在 onStartCommand()方法中创建和运行一个新线程，在后台执行处理，并在该线程完成后终止这个 Service。

3. 启动和停止 Service

Service 不能自己运行，需要通过调用 Context.startService()或 Context.bindService()方法启动服务。和启动一个 Activity 一样，可以在注册了合适的 Intent Receiver 后隐式地启动 Service，也可以显式地指定 Service 的类名来启动 Service。后者在 Activity 中通过调用 context.startService()来启动 Servic，它可以传递参数给 Service。Service 一般是依次调用 onCreate()和 onStartCommand()方法完成启动过程。当 Service 需要停止时，一般是调用 stopService()方法结束，Service 将会调用 onDestroy()方法销毁它。

对于 Service 的启动和停止过程是不能嵌套的。无论 startService()方法被调用了多少次，只需调用一次 stopService()方法就会停止 Service。

由于 Service 具有高优先级，它们通常不会被运行时终止，因此当 Service 的处理完成后，应该调用 stopSelf()方法显式地终止它，这样可以避免系统仍然为该 Service 保留资源，改善应用程序中的资源占用情况。

【例 7-1】 工程 07_ServiceExample 演示了 Service 创建、启动的方法。工程中用到的部件有 3 个。

(1) MediaPlayer 类。它是一个媒体播放器类，可以播放音频和视频文件。

MediaPlayer 类的详细用法将在第 9 章介绍。

（2）Intent。一般是用作 Activity、Service、BroadcastReceiver 之间传递数据，以及启动。

（3）Pendingintent。一般用在 Notification 上，可以理解为延迟执行的 Intent，PendingIntent 是对 Intent 的一个包装，可以把这个描述交给其他应用程序，该程序根据这个描述在后面的时间执行安排好的操作。

示例工程的实现过程如下。

步骤 1：创建工程 07_ServiceExample。创建一个新的 Java 类文件作为 Service，类名为 MyMusicService，继承自 Android. app. Service，本例 Service 类负责播放一个音乐文件。这段代码的主要工作是重写 onCreate()、onStartCommand()、onDestroy()等方法。程序设计者需要这个 Service 完成的功能在其 onCreate()和 onStartCommand()中实现即可。

因为这个音乐播放器是在后台运行的，为了在手机上获得当前状态，本例使用 Notification 来显示 Service 的当前状态。实际运行效果如图 7-2 所示。

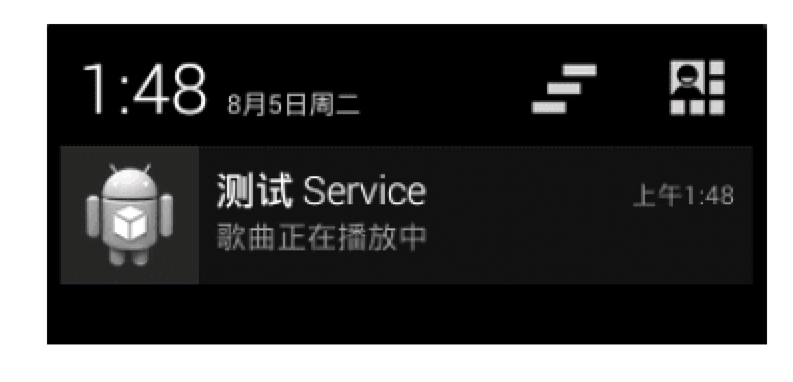


图 7-2　使用 Notification 显示 Service 的状态

MyMusicService 类的主要代码如下：

```
//package 和 import 语句略
public class MyMusicService extends Service {
                //注意：这里是继承自 Service，不是通常的 Activity
    private NotificationManager mynm;
    private MediaPlayer myplayer;              //定义播放器实例 myplayer
    public void onCreate() {                   //这个方法会在 Service 创建时被调用
        super.onCreate();
        myplayer=MediaPlayer.create(this, R.raw.music01);
                //用 MediaPlayer 来播放指定路径下的文件
        mynm= (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    }
    public void onStartCommand(Intent intent, int flags,int startId) {
                //该方法在每一次 startService 时被调用。
        myplayer.start();                      //启动播放音乐
        Notification notification= new Notification(R.drawable.ic_launcher, "Service started",
        System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new Intent(this,
        MainActivity.class), 0);
        notification.setLatestEventInfo(this, "测试Service","歌曲正在播放中", contentIntent);
        mynm.notify("Hello", 0, notification);
    }
    public void onDestroy() {                  //该方法会在 Service 销毁时被调用
        myplayer.stop();                       //停止播放器播放音乐
        Notification notification= new Notification(R.drawable.ic_launcher, "Service stopped",
        System.currentTimeMillis());
```




```
        PendingIntent contentIntent = PendingIntent. getActivity (this, 0, new Intent (this,
        MainActivity.class), 0);
        notification.setLatestEventInfo(this, "Service结束","歌曲播放停止!", contentIntent);
                                                //通知信息
        mynm.notify("Hello", 0, notification);
        super.onDestroy();                      //销毁 Service
    }
}
```

步骤 2：本例是用一个 Activity 来显式地启动一个 Service，因此需要设计一个启动这个 Service 的 Activity 类。首先设计 Activity 的布局。在布局 XML 文件中添加两个按钮用来启动、停止 Service。

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Service创建、启动、停止示例:\n"  />
    <Button android:id="@+id/btn_start"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="启动 Service"  />
    <Button android:id="@+id/btn_stop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="停止 Service"  />
</LinearLayout>
```

步骤 3：设计启动 Service 的 Activity，类名为 MainActivity。本例中是通过 startService()方法启动 Service。主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    Button btnStart,btnStop;                    //定义两个按钮
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnStart=(Button)findViewById(R.id.btn_start);
        btnStop=(Button)findViewById(R.id.btn_stop);
        btnStart.setOnClickListener(listen_btnStart);
```

```
        btnStop.setOnClickListener(listen_btnStop);
    }
    private OnClickListener listen_btnStart=new OnClickListener() {
            public void onClick(View v) {
            startService(new Intent(MainActivity.this, MyMusicService.class));
             //启动 Service,第一个参数是源 Activity名,第二个参数是要启动的 Service名
        }
    };
    private OnClickListener listen_btnStop=new OnClickListener() {
        public void onClick(View v) {
            stopService(new Intent(MainActivity.this,MyMusicService.class));
              //终止 Service,第一个参数是源 Activity名,第二个参数是要终止的 Service名
        }
    };
}
```

步骤 4：在 AndroidManifest.xml 配置文件中添加对 MyMusicService 的声明，使其可以被访问。声明写在</application>标签之前。

```
<service
    android:enabled="true"
    android:name=".MyMusicService" />
```

示例工程中的 MainActivity 界面如图 7-3 所示。



图 7-3 MainActivity 界面

7.2.2 将 Service 绑定到 Activity

在 Activity 中通过调用 context.bindService()方式来启动并绑定一个 Service。要让一个 Service 支持绑定，需要实现并重写 Service 的 onBind()方法，该方法要求返回被绑定 Service 的当前实例。例如：

```
public class MyMusicService extends Service {
    @Override
    public IBinder onBind(Intent intent)  {        //成功绑定后调用该方法
        return mybinder;
    }
    public class MyBinder2 extends Binder{
        MyService getService() {
            return MyMusicService.this;
        }
    }
    private final IBinder mybinder=new MyBinder2();
}
```

Service 和其他组件之间的连接表示为一个 ServiceConnection。要想将一个 Service

和其他组件进行绑定，需要实现一个新的 ServiceConnection，建立了一个连接之后，就可以通过重写 onServiceConnected()和 onServiceDisconnected()方法来获得对 Service 实例的引用。例如：

```
private MyService myService;
private ServiceConnection serviceConnection= new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        //成功连接服务后，该方法被调用。在该方法中可以获得 MyService 对象
        myService= ((MyService.MyBinder) service).getService();
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        //连接服务失败或 Service 意外断开后，该方法被调用
        myService= null;
    }
};
```

要执行绑定，需要在 Activity 中调用 bindService()方法，方法的调用格式如下：

```
bindService(Intent service, ServiceConnection conn, int flags)
```

需要传递给它 3 个参数，第 1 个参数是要绑定的 Service 的 Intent，第 2 个参数是一个 ServiceConnection 实现的实例，第 3 个参数是绑定标识，通常使用系统定义的常量。

一旦 Service 被绑定，就可以通过从 onServiceConnected()处理程序获得的 serviceBinder 对象来使用 Service 所有的公共方法和属性。

当通过 bindService()方法被成功绑定后，Service 一般是依次调用 onCreate()和 onBind()方法。当通过 unbindService()方法结束时，Service 会依次调用 unbind()和 onDestroy()方法。通过 bindService()方法，Service 就和调用 bindService()的进程绑定了，当调用 bindService()的进程结束后，其绑定的 Service 也就被结束了。这一点是和 startService()不一样的地方。

【例 7-2】 工程 07_BindServiceExample 演示了 Service 绑定和解除绑定的方法。

示例工程的实现过程如下。

步骤 1：创建工程 07_BindServiceExample。在布局 XML 文件中添加两个按钮用来绑定、解除绑定 Service。XML 文件内容与例 7-1 类似，在此不再赘述。

步骤 2：创建一个新的 MyService 类，该类继承自 Android. app. Service。在该类中重写了几个与绑定相关的方法。

在 MyService 类中定义了一个 MyBinder 类，用于获得 MyService 的对象实例。在 ServiceConnection 接口的 onServiceConnected 方法中的第 2 个参数是一个 IBinder 类型的变量，将该参数转换成 MyService. MyBinder 对象，并使用 MyBinder 类中的 getService 方法获得 MyService 对象。在获得 MyService 对象后，就可以在 Activity 中操作

MyService。

```
//package 和 import 语句略
public class MyService extends Service {
    private static final String TAG= "MyService";
    private NotificationManager mynm;
    private MyBinder myBinder= new MyBinder();

    @Override
    public void onCreate() {                    //这个方法在 Service 创建时被调用
        Log.d(TAG, "MyService:onCreate()被执行");
        super.onCreate();
        mynm= (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
        Notification notification = new Notification (R.drawable.ic_launcher, "Service started",
        System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent. getActivity (this, 0, new Intent (this,
        MainActivity.class), 0);
        notification.setLatestEventInfo(this, "测试Service","Service被绑定", contentIntent); //通知信息
        notification.flags= notification.flags|Notification.FLAG_AUTO_CANCEL;
                                                  //设置 notification 被单击后自动取消自己
        mynm.notify("Hello", 0, notification);
    }

    @Override
    public IBinder onBind(Intent intent) {      //成功绑定后调用该方法
        Log.d(TAG, "MyService:onBind()被执行");
        return myBinder;
    }

    @Override
    public boolean onUnbind(Intent intent) {    //解除绑定时调用该方法
        Log.d(TAG, "MyService:onUnbind()被执行");
        mynm= (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
        Notification notification = new Notification (R.drawable.ic_launcher, "Service stopped",
        System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent. getActivity (this, 0, new Intent (this,
        MainActivity.class), 0);
        notification. setLatestEventInfo (this, "测试 Service"," Service 被解除绑定",
        contentIntent);                         //通知信息
        notification.flags= notification.flags|Notification.FLAG_AUTO_CANCEL;
                //设置 notification 被单击后自动取消自己
        mynm.notify("Hello", 0, notification);
        return super.onUnbind(intent);
    }
```




```
    @Override
    public void onDestroy() {
        Log.d(TAG, "MyService:onDestroy()被执行");
        super.onDestroy();
    }

    public class MyBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
}
```

步骤 3：设计绑定 Service 的 Activity，类名为 MainActivity。定义一个 MyService 变量和一个 ServiceConnection 变量，主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private MyService myService;
    private ServiceConnection serviceConnection=new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
              //成功连接服务后,该方法被调用。在该方法中可以获得 MyService 对象
            myService=((MyService.MyBinder)service).getService();
            Toast.makeText(MainActivity.this,"Service  Connected.", Toast.LENGTH_LONG).show();
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
              //连接服务失败后,该方法被调用
            myService=null;
            Toast.makeText(MainActivity.this, "Service  Failed.", Toast.LENGTH_LONG).show();
        }
  };
    Button btnStart,btnStop;                    //定义两个按钮
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnStart=(Button)findViewById(R.id.btn_start);
        btnStop=(Button)findViewById(R.id.btn_stop);
        btnStart.setOnClickListener(listen_btnStart);
        btnStop.setOnClickListener(listen_btnStop);
    }
    private OnClickListener listen_btnStart=new OnClickListener() {
```

```
            public void onClick(View v) {
                Intent serviceIntent=new Intent(MainActivity.this, MyService.class);
                bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
                                //绑定 Service
            }
        };
        private OnClickListener listen_btnStop=new OnClickListener() {
            public void onClick(View v) {
                unbindService(serviceConnection);
                                //解除绑定 Service
            }
        };
    }
```

步骤 4：在 AndroidManifest.xml 中添加对 MyService 服务的声明，使其可以被访问。

```
<service
    android:enabled="true"
    android:name=".MyService" />
```

运行程序后，单击按钮绑定 Service 和解除绑定时的 Notification 提示信息如图 7-4 所示。



(a) (b)

图 7-4 Notification 提示信息

7.2.3 创建前台 Service

Android 系统采用一种动态的方法管理资源，应用程序可能会在没有警告的情况下被终止。当系统资源紧张，确定哪个应用程序或组件可以被终止时，Android 给正在运行的 Service 赋予了第二高的优先级，只有处于激活状态、前台运行的 Activity 才可以拥有最高的优先级。

有时需要把 Service 的优先级提升到与前台 Activity 一样高，这样有利于有些 Service 直接和用户交互。这可以通过调用 Service 的 startForeground()方法将 Service 设置为前台运行。

由于前台 Service 需要直接和用户交互，因此在调用 startForeground()方法时，必须指定一个持续工作的 Notification，只要 Service 在前台运行，这个通知就会显示。同时还

要为用户提供一个禁用前台 Service 的方式，通常的方式是通过单击正在运行的 Notification 打开一个 Activity。

以下是一个将 Service 移至前台的代码片段：

```
private void startPlayback(String album,String artist) {
    int NOTIFICATION_ID=1;
    Intent intent=new Intent(this,MyActivity.class);
        //创建一个当单击通知时打开 MyActivity 的 Intent
    PendingIntent pi=PendingIntent.getActivity(this, 1, intent, 0);
    Notification notification=new Notification(R.drawable.ic_launcher,"单击取消前台运行",System.
    currentTimeMillis());
        //设置 Notification 的参数
    notification.flags=notification.flags|Notification.FLAG_ONGOING_EVENT;
        //设置 Notification 为持续显示
    startForeground(NOTIFICATION_ID,notification);
        //将 Service 移至前台
}
```

当 Service 不再需要前台运行的优先级时，可以调用 stopForeground()方法来将其移至后台，通常会同时移除 Notification 通知。

需要注意的是，将一个 Service 设为前台运行可以有效地避免运行时被系统终止，但如果同时运行多个这样不可终止的 Service，将会使系统很难从资源缺乏的状态下恢复正常运行。所以，只有有助于 Service 正确运行，并且确实有必要时才应该使用这种技术。

7.2.4　IntentService

应用程序中的 Activity 和 Service 都是运行在主线程中的，当一个 Activity 启动了 Service 后，Service 对象会调用 onStartCommand()方法，如果该方法需要较长时间才能返回，那么主线程中的 Activity 视图对象就会得不到及时刷新，用户就会觉得程序被挂起。所以应当避免在 onStartCommand()方法中执行非常耗时的操作。

当 Service 中要处理较为耗时的操作时，使用 IntentService 是解决上述问题的一个方法。IntentService 是 Service 的子类，它提供了 onHandleIntent()方法。

IntentService 类在创建 Service 时会单独启动一个工作线程(work thread)，并在工作线程中调用 onHandleIntent()方法。因此用户在创建 IntentService 类的子类时，不需要重写 onStartCommand()方法，而是要重写 onHandleIntent()方法完成 Service 的功能。这样，用户就可以将比较耗时的操作放在 onHandleIntent()方法中处理，系统会协调主线程和工作线程，使 Service 和 Activity 都有机会使用 CPU 资源。

另外一点与 Service 不同的是，当 IntentService 对象执行完 onHandleIntent()方法返回时，系统会直接将该 Service 结束以释放资源。

以下是创建 IntentService 的代码框架：

```
public class MyIntentService extends IntentService{
```

```
    public MyIntentService(String name) {
        super(name);
    }

    public void onCreate() {
        super.onCreate();
    }

    protected void onHandleIntent(Intent arg0) {
        …                                       //Service的操作任务
    }
}
```

与启动 Service 类似，调用 startService()方法可以启动 IntentService。例如，在 Activity 中使用如下代码启动 IntentService：

```
Intent intent=new Intent(this,MyIntentService.class);
startService(intent);
```

7.3 获得系统服务

7.3.1 系统服务简介

Android 系统提供了很多预置的服务，此类服务称为“系统服务”。系统服务实际上可以看作是一个对象，通过 Activity 类的 getSystemService()方法可以获得指定的系统服务。getSystemService()方法只有一个 String 类型的参数，表示系统服务的 ID，这个 ID 在整个 Android 系统中是唯一的。

例如，获得剪贴板服务(android.text.ClipboardManager 对象)可以用如下代码实现：

```
android.text.ClipboardManager clipboardManager=
(android.text.ClipboardManager)getSystemService(Context.CLIPBOARD_SERVICE);
clipboardManager.setText("这是我设置的剪贴板内容");
```

获得了剪贴板服务对象后，可以调用 ClipboardManager.setText 方法对剪贴板设置文本。之后，在 Android 系统中所有的文本输入框都可以从这个剪贴板对象中获得这段文本。

为了便于记忆和管理，Android SDK 在 android.content.Context 类中定义了如下系统服务的 4 个 ID 常量。

(1) AUDIO_SERVICE：音频服务。

(2) WINDOW_SERVICE：窗口服务。

(3) NOTIFICATION_SERVICE：通知服务。

(4) ALARM_SERVICE：闹钟服务。

本节以 ALARM_SERVICE 为例介绍系统服务的获取和使用方法。

7.3.2　AlarmManager 简介

闹钟应用程序是人们常用的基本应用程序之一。在 Android 系统中闹钟服务功能不仅仅对闹钟应用程序服务,还可以利用闹钟服务功能制作定时器。这样即便应用程序没有运行或者是没有启动的情况下,只要其注册过闹钟,那么该闹钟预定的时间到了之后,Android 系统可以自动将该应用程序启动。

Android 系统提供了 Android.app.AlarmManager 类,用于访问系统定时服务的管理器,开发人员可以在程序中设置某个应用程序在未来的某个时刻被执行。当 AlarmManager 定时时间到了之后,已注册的 Intent 对象将会被系统广播,进而启动目标程序。

1. AlarmManager 的常用方法

AlarmManager 的常用方法有 3 个。

1) set(int type,long triggerAtMillis,PendingIntent operation)

该方法用于设置一次性闹钟,第一个参数表示闹钟类型,第二个参数表示闹钟执行时间,第三个参数表示闹钟响应动作。

2) setRepeating(int type,long triggerAtMillis,long intervalMillis,PendingIntent operation)

该方法用于设置重复闹钟,第一个参数表示闹钟类型,第二个参数表示闹钟首次执行时间,第三个参数表示闹钟两次执行的间隔时间,第四个参数表示闹钟响应动作。

3) setInexactRepeating(int type,long triggerAtMillis,long intervalMillis,PendingIntent operation)

该方法也用于设置重复闹钟,与第二个方法相似,但是两次闹钟执行的间隔时间不是固定的。

2. AlarmManager 方法的参数

这 3 个方法使用的参数基本相同,分别如下。

1) int type

闹钟的类型。

2) long triggerAtMillis

闹钟的第一次执行时间,以毫秒为单位,一般使用当前时间,也可以是自定义时间。需要注意的是,该参数与第一个参数(type)密切相关。如果第一个参数对应的闹钟使用的是相对时间(ELAPSED_REALTIME 和 ELAPSED_REALTIME_WAKEUP),那么该参数就必须使用相对时间,当前时间表示为 SystemClock.elapsedRealtime()。如果第一个参数对应的闹钟使用的是绝对时间(RTC、RTC_WAKEUP、POWER_OFF_WAKEUP),那么该参数就必须使用绝对时间,当前时间表示为 System.currentTimeMillis()。

3) long intervalMillis

当设置为重复闹钟时,该参数表示两次闹钟执行的间隔时间,以毫秒为单位。

4) PendingIntent operation

是闹钟的执行动作,例如发送一个广播、给出提示等。

3. 闹钟的类型

在 Android 系统中,AlarmManager 提供了 5 种类型的闹钟服务。

1) AlarmManager. ELAPSED_REALTIME

当系统进入睡眠状态时,这种类型的闹钟不会唤醒系统。直到系统下次被唤醒才传递它,该闹钟所用的时间是相对时间,是从系统启动后开始计时的,计时包括睡眠时间。可以通过调用 SystemClock. elapsedRealtime()获得该时间。

2) AlarmManager. ELAPSED_REALTIME_WAKEUP

当系统进入睡眠状态时,这种类型的闹钟能唤醒系统并执行提示功能,其余用法与 ELAPSED_REALTIME 相同,闹钟也使用相对时间。

3) AlarmManager. RTC

当系统进入睡眠状态时,这种类型的闹钟不会唤醒系统。直到系统下次被唤醒才传递它,该闹钟所用的时间是绝对时间,即当前系统时间。

4) AlarmManager. RTC_WAKEUP

当系统进入睡眠状态时,这种类型的闹钟能唤醒系统并执行提示功能,其余用法同 RTC 类型相同,闹钟使用绝对时间。

5) AlarmManager. POWER_OFF_WAKEUP

这种类型的闹钟在设备关机状态下也能正常进行提示功能,该状态下闹钟也使用绝对时间。

7.3.3 PendingIntent

在设置闹钟时,使用了 PendingIntent。PendingIntent 是 Intent 的进一步封装,它既包含 Intent 的描述,又是 Intent 行为的执行。可以通过调用 getActivity()、getBroadcast()或 getService()方法来得到一个 PendingIntent 实例。

1) public static PendingIntent getBroadcast(Context context, int requestCode, Intent intent, int flags)

通过该方法获得的 PendingIntent 将会扮演一个广播的功能,类似于调用 Context. sendBroadcast()方法。当系统通过它要发送一个 Intent 时要采用广播的形式,并且在该 Intent 中会包含相应的 Intent 接收对象,这个对象可以在创建 PendingIntent 的时候指定,也可以通过 ACTION 和 CATEGORY 等描述让系统自动找到该行为处理对象。

第二个参数 requestCode 一定要是唯一的,如果系统需要多个定时器的话,必须使用不同的 ID。

2) Public static PendingIntent getActivity((Context context, int requestCode, Intent intent, int flags)

通过该方法获得的 PendingIntent 可以直接启动新的 Activity,类似于调用 Context.

startActivity(Intent)。但需要注意要想这个新的 Activity 不再是当前进程存在的 Activity 时。在 Intent 中必须使用 Intent.FLAG_ACTIVITY_NEW_TASK.

3) public static PendingIntent getService(Context context, int requestCode, Intent intent, int flags)

通过该方法获得的 PengdingIntent 可以直接启动新的 Service,类似于调用 Context.startService()方法。

需要注意的是,如果是通过启动服务来实现闹钟提示的话,PendingIntent 对象的获取就应该采用 Pending.getService()方法;如果是通过广播来实现闹钟提示的话,PendingIntent 对象的获取就应该采用 PendingIntent.getBroadcast()方法;如果是采用 Activity 的方式来实现闹钟提示的话,PendingIntent 对象的获取就应该采用 PendingIntent.getActivity()方法。

以通过广播来实现闹钟提示为例,AlarmManager 的一般应用方法如下。

首先定义和声明一个 BroadcastReceiver 对象,用于接收闹钟事件的广播消息。该接收对象可以在 AndroidManifest.xml 中静态注册,也可以在 Java 代码中动态注册。例如,在 AndroidManifest.xml 中注册:

```
<receiver
    android:name=".ClockAlarmReceiver">
    <intent-filter>
        <action android:name="com.android.alarmclock.ALARM_ALERT"/>
    </intent-filter>
</receiver>
```

然后创建一个 PendingIntent:

```
Intent intent=new Intent(ALARM_ALERT_ACTION);
intent.putExtra(ID, id);
intent.putExtra(TIME, atTimeInMillis);
PendingIntent sender=PendingIntent.getBroadcast(context, 0 , intent, PendingIntent.FLAG_CANCEL_
CURRENT);
```

通过 Context 对象的 getSystemService(Context.ALARM_SERVICE)方法可以获得 AlarmManager 对象。如果设置为一次性闹钟,则调用 set()方法,例如:

```
AlarmManager am=(AlarmManager)getSystemService(ALARM_SERVICE);
am.set(AlarmManager.POWER_OFF_WAKEUP, atTimeInMillis, sender);
```

如果设置为重复性闹钟,则调用 setRepeating()方法。例如,设置重复间隔 30s 的闹钟:

```
long firstTime=System.currentTimeMillis();
AlarmManager am=(AlarmManager)getSystemService(ALARM_SERVICE);
am.setRepeating(AlarmManager.POWER_OFF_WAKEUP, firstTime, 30 * 1000 , sender);
```

7.3.4 使用系统闹钟服务

【例 7-3】 示例工程 07_AlarmManagerExample 程序中使用时间选择对话框设置闹

钟的时间，预设时间到后弹出提醒对话框。

本例定义了 3 个 Java 类，分别是 MainActivity、ClockAlarmReceiver 和 ClockAlarmActivity。在 MainActivity 中使用时间选择对话框设置闹钟的时间。当 AlarmManager 定时时间到了之后，注册的 Intent 对象将会被系统广播。ClockAlarmReceiver 接收到系统广播后，进而启动 ClockAlarmActivity，弹出提醒对话框。

示例工程的实现过程如下。

步骤 1：新建工程项目 07_AlarmManagerExample，定义 activity_main.xml 文件。该文件定义了 MainActivity 的界面布局。布局包括提示文字和一个按钮。用户单击这个按钮，就会弹出“时间选择”对话框，设置闹钟时间。MainActivity 的界面布局如图 7-5 所示。

步骤 2：定义 MainActivity 类。主要设计工作有重写 onCreate()方法和 onDestroy()方法、广播接收器类对象的实例化、按钮事件单击响应的方法等。

单击按钮后，弹出“时间选择”对话框，设置闹钟时间。该功能通过实例化 android.app.TimePicker Dialog 类实现，通过 Calendar 类得到系统的当前时间。运行结果如图 7-6 所示。



图 7-5　MainActivity 的界面布局



图 7-6　设置闹钟时间

闹钟时间设置完成后，使用 Toast 显示一个提示信息，如图 7-7 所示。

闹钟设置完成
设置时间：9:42

图 7-7　闹钟时间设置完成后的提示信息

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private Button btn=null;
    private AlarmManager alarmManager=null;
    Calendar cal=Calendar.getInstance();
```

```
final int DIALOG_TIME=0;                                        //设置对话框 id
ClockAlarmReceiver myReceiver=new ClockAlarmReceiver();         //实例化接收器
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    alarmManager=(AlarmManager)getSystemService(Context.ALARM_SERVICE);
    btn=(Button)findViewById(R.id.btn);
    btn.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            showDialog(DIALOG_TIME);                            //显示时间选择对话框
        }
    });
}
@Override
protected Dialog onCreateDialog(int id) {
    Dialog dialog=null;
    switch(id) {
    case DIALOG_TIME:
      dialog=new TimePickerDialog(this,
        new TimePickerDialog.OnTimeSetListener() {
          public void onTimeSet(TimePicker timePicker, int hourOfDay,int minute) {
            Calendar c=Calendar.getInstance();                  //获取日期对象
            c.setTimeInMillis(System.currentTimeMillis());      //设置 Calendar 对象
            c.set(Calendar.HOUR, hourOfDay);                    //设置闹钟的小时数
            c.set(Calendar.MINUTE, minute);                     //设置闹钟的分钟数
            c.set(Calendar.SECOND, 0);                          //设置闹钟的秒数
            c.set(Calendar.MILLISECOND, 0);                     //设置闹钟的毫秒数
            Intent intent=new Intent("edu.hebust.zxm.ALARM_ALERT_ACTION");
                        //创建 Intent 对象
            PendingIntent sender=PendingIntent.getBroadcast(MainActivity.this, 0, intent,
            PendingIntent.FLAG_CANCEL_CURRENT);
                        //创建 PendingIntent 对象
            AlarmManager am=(AlarmManager)getSystemService(ALARM_SERVICE);
                        //获取 AlarmManager 对象
            am.set(AlarmManager.RTC_WAKEUP, c.getTimeInMillis(), sender);
                       //设置闹钟
            Toast.makeText(MainActivity.this, "闹钟设置完成\n设置时间:"+String.valueOf
            (c.get(Calendar.HOUR))+":"+String.valueOf(c.get(Calendar.MINUTE)), Toast.LENGTH_
            LONG).show();                                               //提示用户
          }
        },cal.get(Calendar.HOUR_OF_DAY),cal.get(Calendar.MINUTE),false);
     break;
   }
 return dialog;
```

```
    }
}
```

步骤 3：定义广播接收器。类名为 ClockAlarmReceiver，它继承自 BroadcastReceiver 类，负责接收到广播信息后启动 ClockAlarmActivity。主要代码如下：

```
//package 和 import 语句略
public class ClockAlarmReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Intent i=new Intent(context, ClockAlarmActivity.class);
        i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(i);
    }
}
```

步骤 4：定义弹出提醒信息的 TimeAlarmActivity，主要功能是弹出提醒对话框，运行结果如图 7-8 所示。



图 7-8　闹钟时间到的提醒信息

TimeAlarmActivity 类的主要代码如下：

```
//package 和 import 语句略
public class ClockAlarmActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.clockalarm);
        new AlertDialog.Builder(ClockAlarmActivity.this)          //显示对话框
            .setTitle("闹钟提醒")                                   //设置标题
            .setMessage("您设置的闹钟时间到了！")                     //设置内容
            .setPositiveButton("知道了", new OnClickListener() {    //设置按钮
```

```
                    public void onClick(DialogInterface dialog, int which) {
                        ClockAlarmActivity.this.finish();           //关闭 Activity
                    }
                })
                .create().show();
        }
}
```

步骤 5：在 AndroidManifest.xml 中声明 Activity 和 BroadcastReceiver 对象，主要代码如下：

```
<activity android:name=".ClockAlarmActivity" />
<receiver
    android:name=".ClockAlarmReceiver"
    android:process=":remote">
    <intent-filter>
        <action android:name="edu.hebust.zxm.ALARM_ALERT_ACTION"/>
    </intent-filter>
</receiver>
```

7.4　综合使用 Service 和 BroadcastReceiver

Service 和 BroadcastReceiver 综合使用，可以使应用程序完成更复杂、更完善的功能。

【例 7-4】　工程 07_ServiceBroadcastExample 演示了一个综合使用 Service 和 BroadcastReceiver 的实例。

示例程序实现了一个音乐播放器，用户可以启动、暂停和停止音乐的播放。程序中使用 Service 在后台播放音乐，而用 Broadcast 发送广播通知 Activity 更改界面。其实现过程如下。

步骤 1：新建一个工程 07_ServiceBroadcastExample。首先设计 Activity 的布局。布局中放置了两个按钮，分别是“播放”按钮和“停止”按钮，当音乐开始播放时，“播放”按钮变为“暂停”按钮，具体如图 7-9 所示。



(a)　　(b)

图 7-9　示例程序的运行界面

步骤 2：设计主 Activity。

在程序 MainActivity 类设置了一个状态变量 state，用于标识当前的播放状态，根据这个状态值来决定左侧按钮的显示文字，以及单击这个按钮之后的处理内容。

在 onCreate()方法中主要完成 BroadcastReceiver 的注册及两个按钮单击响应。本例中采用动态注册方法 registerReceiver()对广播进行注册。另外，在 MainActivity 类中还进行了 BroadcastReceiver 类对象的实例化。BroadcastReceiver 对象主要负责监听 Service 发送的广播及接收音乐状态值，然后修改 Activity 中的提示文字和按钮上的文字。

相关代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private TextView tv;
    private Button play, stop;
    public static int state= 3;              //状态变量:1表示"播放",2表示"暂停",3表示"停止"

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.setTitle("综合应用:Service 和 BroadcastReceiver");
        tv= (TextView)findViewById(R.id.text);
        play= (Button)findViewById(R.id.play);
        stop= (Button)findViewById(R.id.stop);
        IntentFilter inf= new IntentFilter();
        inf.addAction("edu.hebust.zxm.MUSIC_ACTION");
        registerReceiver(broad, inf);
                //注册 BroadcastReceiver
        play.setOnClickListener(new OnClickListener() {          //播放按钮响应
            public void onClick(View v) {
                switch(state) {
                case 1:
                    state= 2;
                    break;
                default:
                    state= 1;
                    break;
                }
                Intent intent=new Intent(MainActivity.this,AudioService.class);
                intent.putExtra("action", state);              //将状态值传给 Service
                startService(intent);                          //启动 Service
            }
        });
        stop.setOnClickListener(new OnClickListener() {          //停止按钮响应
            public void onClick(View v) {
```

```
                Intent intent=new Intent(MainActivity.this,AudioService.class);
                stopService(intent);                         //停止 Service
            }
        });
    }

    //实例化 BroadcastReceiver 对象
    public BroadcastReceiver broad=new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            int i=intent.getIntExtra("action",-1);               //获得 Service 发送的状态
            switch(i) {
            case 1:                                          //播放
                play.setText("暂停");
                tv.setText("音乐文件正在播放……\n");
                break;
            case 2:                                          //暂停
                play.setText("播放");
                tv.setText("音乐文件正在暂停……\n");
                break;
            default:                                         //停止
                play.setText("播放");
                tv.setText("音乐文件通知播放……\n");
                if(3==i) {
                    state=3;
                }
                break;
            }
        }
    };

    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(broad);
    }
}
```

当单击“播放”按钮后，改变音乐播放器的状态值，并将其封装在 intent 中，之后调用 startService(intent)方法启动服务，并将状态值传递给 Service。Service 将会根据接收到的状态值进行不同的处理。

当单击“停止”按钮后，调用 stopService()方法停止 Service，音乐停止播放。

在 onDestroy()方法里解除广播的注册，用 unregisterReceiver()方法解除 BroadcastReceiver 的注册。

步骤 3：编写播放器的 AudioService 类，完成 Service 的功能。

在 AudioService 类中声明一个 MediaPlayer 对象，用于播放音乐文件。MediaPlayer 的用法在第 9 章详细介绍。

单击按钮开启服务或停止服务，在后台对 MediaPlayer 对象进行操作，以达到播放、暂停、停止的效果。

```
//package 和 import 语句略
public class AudioService extends Service {
    private MediaPlayer mp;                                         //音乐播放器
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        int i=intent.getIntExtra("action", 0);                    //获得从 activity 传过来的状态值
        if(i==1) {                                                 //i=1 表示播放音乐
            if(null==mp) {
                mp=MediaPlayer.create(this, R.raw.music01);        //实例化 MediaPlayer
            }
            mp.start();                                            //播放音乐
            mp.setOnCompletionListener(new OnCompletionListener() {
                public void onCompletion(MediaPlayer mp) {
                    stopSelf();                                    //音乐播放完，停止服务
                }
            });
        } else if(i==2) {                                          //i=2 表示暂停播放音乐
            if(mp != null && mp.isPlaying()) {
                mp.pause();                                        //暂停播放
            }
        }
        Intent in=new Intent("edu.hebust.zxm.MUSIC_ACTION ");
        in.putExtra("action", i);                                  //将状态传给广播接收者
        sendBroadcast(in);                                          //发送广播，由 Receiver 接收
        return super.onStartCommand(intent, flags, startId);
    }

    public void onDestroy() {
        super.onDestroy();
        mp.stop();                                                 //停止播放器播放
        Intent in=new Intent("edu.hebust.zxm.MUSIC_ACTION ");
        in.putExtra("action", 3);                                  //将状态传给广播接收者
        sendBroadcast(in);                                          //发送广播，由 Receiver 接收
    }
}
```

程序运行后，单击“播放”按钮，就会调用 StartService()方法启动服务，Service 会调用 onStartCommand()方法，在方法中判断音乐播放器的状态值，根据状态值执行播放或暂停，并发送携带状态值的广播，BroadcastReceiver 接收到广播信息后，根据接收到的值

修改 Activity 界面中显示的文字。

关闭服务时会自动调用 Service 的 onDestroy()方法，在此方法中调用 MediaPlayer 对象的 stop()方法停止播放器的播放，并向 MainActivity 的广播接收器发送停止播放的状态值 3，让 MainActivity 改变播放按钮的显示。

这个示例程序结合了 Service 和 BroadcastReceiver 两部分的内容，Service 负责在后台执行音乐的播放和停止，而 BroadcastReceiver 负责前台显示的改变，两者结合使得这个音乐播放器的功能更加完善。

7.5　本章小结

本章学习了 Service 和 Broadcast 的应用。Service 是 Android 系统中 4 个应用程序组件之一。通过启动一个 Service，可以在不显示界面的前提下在后台运行指定的任务，这样可以不影响用户正在进行的其他操作。这也是 Service 的重要用途之一。Service 与 Activity 的作用相近，但是不能自己运行，需要通过某一个 Activity 或者其他 Context 对象来调用 startService()方法或 bindService()方法来启动。学习本章的重点是 Service 的设计实现以及启动、停止的方法。

习　　题

1. 什么是 Service？ Service 与 Broadcast 有什么不同？

2. 调用 startService()和 bindService()方法启动服务有什么区别？

3. 在 Service 对象的生命周期内，Service 对象会多次调用 onCreate()方法吗？ 会多次调用 onStartCommand()方法吗？

4. IntentService 类创建的 Service 对象有什么特点？

5. 使用 Service 实现一个音乐播放盒程序，当用户单击“播放”按钮，即使退出当前操作进行其他操作，音乐也不会停止播放。通过菜单选项退出音乐播放器，并在退出前弹出一个“提示”对话框，要求用户进行确认。

6. 运行示例工程 07_BindServiceExample，观察其 LogCat 输出信息，理解绑定 Service 时相关方法的调用和执行过程。



数据的存储和访问

在移动设备的使用过程中，经常会遇到一些数据（如照片、视频、电话号码和备忘录等）需要永久存储。这些数据不能因为关机或重启而丢失，而且经常需要访问，访问方式包括读取、修改、插入和删除等。Android 系统提供了多种数据存储和访问方式，本章介绍其中常用的文件存储、SQLite 数据库存储和内容提供器（Content Provider）方式。其他还有 SharedPreferences 方式和网络存储方式等，限于篇幅不再介绍，具体使用方法请读者参阅相关文献。

8.1 数据文件的存储和访问

Android 使用的是基于 Linux 的文件系统，开发人员可以建立和访问程序自身的私有文件，也可以访问保存在资源目录中的数据文件和 XML 文件，还可以访问 SD 卡等外部存储设备中的文件。

8.1.1 数据文件的存取操作

当 Android 的应用程序被安装后，其所在的安装包中会有一个相应的文件夹用于存放自己的数据，其路径是/data/data/<package name>/files/。应用程序自己对这个文件夹有写入权限，可以创建文件并存储在这个文件夹中，其他应用程序不能访问它们。当用户卸载应用程序时，其创建的文件也一并被删除。

Context 类中提供了两个方法，即 openFileOutput()和 openFileInput()，可以直接读写文件中的数据，得到文件输出流和输入流。

（1）得到文件输出流，向文件中写入数据。

向文件中写入数据，需要首先调用 openFileOutput()方法得到文件输出流对象，方法的定义如下：

```
public FileOutputStream openFileOutput(String name, int mode)
```

该方法为写入数据做准备而打开文件，如果指定的文件不存在，则自动创建一个新的文件。函数的返回值是 FileOutputStream 类型的对象。

第 1 个参数是准备写入数据的文件名，文件名中不能包含路径分隔符/，创建的文件一般保存在/data/data/<package name>/files 目录中。



第 2 个参数指定了文件的操作模式，可供选择的模式有以下 4 种。

① MODE_APPEND：如果文件已经存在，则在文件数据后添加数据，否则创建文件。

② MODE_PRIVATE：是默认的文件操作方式，这种方式下写入的数据将覆盖原数据。如果文件不存在，则创建文件。

③ MODE_WORLD_READABLE：允许其他应用读取此文件。

④ MODE_WORLD_WRITEABLE：允许其他应用写入此文件。

如果想要具有多个权限时，操作模式之间用"＋"分开。例如，如果想同时得到读与写的权限，则可以通过"MODE_WORLD_READABLE＋MODE_WORLD_WRITEABLE"的方式创建一个 mode。

在进行文件写入操作时，Activity 通过 openFileOutput()方法获得标准数据输出流对象，然后调用该对象的 write()方法将数据写入，最后调用 close()方法关闭输出流。

(2) 得到文件输入流，从文件中读取数据。

与写文件类似，从文件中读取数据，需要首先调用 openFileInput()方法得到文件输入流对象，该方法的定义如下：

```
public FileInputStream openFileInput(String name)
```

该方法为读取数据做准备而打开一个文件，方法的返回值是 FileInputStream 类型。调用该方法就会打开/data/data/<package name>/files 目录中存放的指定文件并读取其中的数据。

方法中的参数是准备读出数据的文件名。同样，文件名不能包含路径分隔符/。调用文件输入流对象的 read()方法将数据读出，最后调用 close()方法关闭输入流。read()方法中的参数是一个字节数组类型，用来存储从输入流中读出的数据。



图 8-1　示例工程 08_FileWriteReadExample 的界面布局

【例 8-1】　示例工程 08_FileWriteReadExample 演示了有关读写文件的方法。Activity 的界面布局如图 8-1 所示，布局中包括 EditText 控件、Button 控件、显示从文件中读出结果的 TextView 控件。

在文本输入框中输入文字后，单击"保存到文件"按钮，文字内容就保存到指定文件，单击"从文件中读取"按钮后，从文件中读取内容，显示到按钮下方的 TextView 控件中。

本例涉及文件的输入输出操作，需要用 import 语句引入 java.io.IOException、java.io.InputStream、java.io.OutputStream、java.io.FileNotFoundException 等相关包文件。MainActivity.java 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
/* 写文件部分 */
    private EditText TxtInput;                    //编辑框,用户输入字符串
    private Button BtnSave,BtnOpen,BtnClean;
    private String Text_of_input;                 //用户输入的字符串
    private OutputStream os;                      //文件输出流
/* 读取部分 */
    private TextView TvShowTxt;                   //TextView,显示读取文件内容
    private String Text_of_read;                  //从文件中读取到的字符串
    private InputStream is;                       //文件输入流
    private byte[] b;                             //字节数组,用来读取文件内容
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TxtInput= (EditText)findViewById(R.id.EditText_Txt);
        BtnSave= (Button)findViewById(R.id.Button_Save);
        BtnOpen= (Button)findViewById(R.id.Button_openTxt);
        BtnClean= (Button)findViewById(R.id.Button_clean);
        TvShowTxt= (TextView)findViewById(R.id.TextView_showTxt);
        /* 以下添加事件的处理方法 */
        BtnSave.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    saveFile();                   //保存文件
                }
        });
        BtnOpen.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    openFile();                   //打开并读取文件
                }
        });
        BtnClean.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    cleanTxt();                   //清空输入框和读取结果
                }
        });
    }
    private void saveFile() {                     //文件保存
        Text_of_input=TxtInput.getText().toString();
                                                  //获得用户输入的字符串
        try {
            os= this.openFileOutput("mytxt", MODE_PRIVATE);
                //打开文件输出流,文件名称为 mytxt,以覆盖模式打开,如果文件不存在则创建
```

```
                文件
            os.write(Text_of_input.getBytes());
                //把字符串转换成字节数组,写入文件中
            } catch(FileNotFoundException e) {
                ...                              //文件未找到,异常
            } catch(IOException e) {
                ...                              //文件写入错误
            } finally {
                try {
                    os.flush();                  //将缓冲区内所有的数据写入文件
                    os.close();                  //关闭文件输出流
                 } catch(IOException e) {
                    ...                          //文件关闭失败
                 }
             }
        Toast.makeText(this,"文本框中的内容已经成功写入文件 mytxt!", Toast.LENGTH_SHORT).show
        ();
    }
    private void openFile() {
        try {
            is= this.openFileInput("mytxt");
                //打开文件输入流,文件名称为 mytxt
            b= new byte[1024];
                //初始化字节数组
            int length= is.read(b);
              //从文件输入流中读取内容到字节数组中
            Text_of_read= new String(b);
              //把字节数组转换成字符串
            setTitle("文件字数:"+ length);       //在 Activity 的标题上显示文件字数
            TvShowTxt.setText(Text_of_read);     //显示文件内容
        } catch(FileNotFoundException e) {
             ...                                 //文件未找到,异常
        } catch(IOException e) {
             ...                                 //文件读取错误,异常
        } finally {
            try {
                is.close();                      //关闭文件输入流
            } catch(IOException e) {
                ...                              //文件关闭失败
            }
        }
    }
    private void cleanTxt() {
        TvShowTxt.setText("");
```

```
            TxtInput.setText("");
        }
    }
```

上例中的文件存放在/data/data/edu. hebust. zxm. filewritereadexample/files/中，文件名为 mytxt，如图 8-2 所示。通过 DDMS 右上方的 pull a file from the device 按钮，可以将这个文件导出到指定的位置并观察其中的内容。

图 8-2　文件的位置

可以将文件写入到 SD 卡中，但需要在 AndroidManifest. xml 配置文件中注册写入外存的权限：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

类似地，可以从 SD 卡的文件中读出数据，但需要在 AndroidManifest. xml 配置文件中注册读外存的权限：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

读写外存时，只需给出文件的完整路径，其余操作与读写内存文件的方法类似。需要注意的是，2. 1 版本以下的 SDCard 路径和 2. 2 之后版本不同，可以通过调用 Environment. getExternalStorageDirectory()方法获取当前 SDCard 位置，这样可以兼容所有版本。另外，可以通过调用 Environment. getExternalStorageState()方法获取 SDCard 的当前状态，常量 Environment. MEDIA_MOUNTED 为 SDCard 已安装状态。

8.1.2 访问资源目录中的数据文件

调用 openRawResource()方法，可以读取资源文件夹 res/raw 中的文件。该方法的参数是要访问文件的资源 ID，方法返回一个 InputStream 类型的对象，可用于读取文件。调用 InputStream 对象的 read()方法可以将数据读出，调用 close()方法则关闭输入流。这种访问只允许读取文件，不能用于更新操作。

【例 8-2】 工程 08_ReadRawExample 演示了读取 res/raw 中数据文件 test. txt 的

内容。

示例程序的功能：单击“读取文件”按钮，将读取的文件内容显示在下方的 TextView 控件中。MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    TextView readTxt;
    String res= "";
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnRead= (Button)findViewById(R.id.Button_Read);
        readTxt= (TextView)findViewById(R.id.readTxt);
        btnRead.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try{
                InputStream in= getResources().openRawResource(R.raw.test);
                        //读取资源文件夹 res/raw 中的文件 test.txt
                int length= in.available();
                        //获取数据流的长度
                byte [] buffer= new byte[length];
                        //定义适当长度的字节数组
                in.read(buffer);
                        //将输入流的数据读入字节数组
                res= EncodingUtils.getString(buffer, "UTF- 8");
                        //将字节数组转换成字符串
                in.close();
                        //关闭输入流
                }catch(Exception e) {
                    e.printStackTrace();
                }
                readTxt.setText(res);              //把得到的内容显示在 TextView 上
            }
        });
    }
}
```

程序的运行结果如图 8-3 所示。

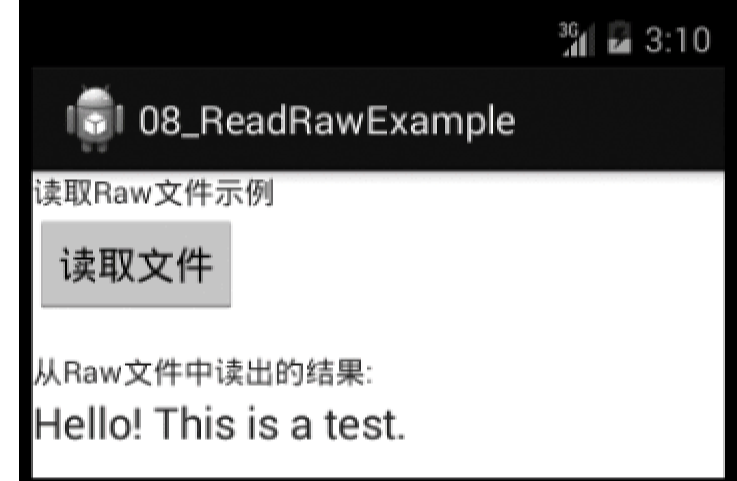


图 8-3　读取资源目录中的数据文件

8.1.3　从 assets 目录中获取文件并读取数据

assets 目录下的文件又称为原生文件，这类文件在被打包成 APK 文件时是不会进行压缩的。Android 系统使用 AssetManager 类实现对

assets 目录下文件的访问，通过 getResources().getAssets()方法可以获得 AssetManager 对象，调用其 open()方法可以根据用户提供的文件名，返回一个 InputStream 对象供用户使用。与调用 openRawResource()方法读取资源文件类似，这种访问只允许读取文件，不能用于更新操作。

【例 8-3】 示例工程 08_ReadAssetsExample 演示了如何读取 assets 目录中的文件。

示例程序的功能：单击"读取文件"按钮，将读取的文件内容显示在下方的 TextView 控件中。响应按钮单击事件的核心代码如下：

```
public void onClick(View v)    {
    try{
        InputStream in=getResources().getAssets().open("test.txt");
                //读取 assets 目录中的文件 test.txt
        int length=in.available();
                //获取数据流的长度
        byte [] buffer=new byte[length];
                //定义适当长度的字节数组
        in.read(buffer);
                //将输入流的数据读入字节数组
        res=EncodingUtils.getString(buffer, "UTF-8");
                //将字节数组转换成字符串
        in.close();
                //关闭输入流
    }catch(Exception e) {
      e.printStackTrace();
     }
    readTxt.setText(res);                          //把得到的内容显示在 TextView 上
}
```

程序的运行结果如图 8-4 所示。



图 8-4 读取 assets 文件夹中的数据文件

8.2 SQLite 数据库的存储和访问

Android 系统集成了 SQLite 嵌入式数据库，每个 Android 应用程序都可以使用。SQLite 是一个比较完整的支持 SQL 的数据库系统。本节主要讲解 SQLite 在 Android 环境中的基本使用。

8.2.1　SQLite 简介

SQLite 是一个非常流行的嵌入式数据库，它是一个免费、开源的数据库系统，具有相当小的内存占用和高速的响应。它能够支持 Windows、Linux、UNIX 等操作系统，支持 Android、Windows Mobile、Symbin 等大部分主流嵌入式操作系统。同时能够与很多编程语言相结合，支持多语言编程接口。

与普通关系数据库一样，SQLite 可以用来存储大量的数据，支持 SQL 查询，能够很容易地对数据进行查询、更新和维护等操作。但由于移动设备平台的内存和外存都受到限制，SQLite 不能执行非常复杂的 Select 语句，不支持外键和左右连接，不支持嵌套事务和部分 ALTER TABLE 功能。

SQLite 是一个轻量级的数据库，它和 C/S 模式的数据库软件不同，是进程内的数据库引擎，因此不存在数据库的客户端和服务器。使用 SQLite 一般只需要带上它的一个动态库，就可以使用它的全部功能。它的核心引擎不依赖第三方的软件，在使用前不需要安装设置，不需要进程来启动、停止或配置，不需要管理员去创建新数据库或分配用户权限，在系统崩溃或失电之后自动恢复。

使用时，访问数据库的程序直接从数据库文件读写，没有中间的服务器进程。而且 SQLite 数据库中所有的信息（表、视图、触发器等）都包含在一个文件内，这个文件可以复制到其他目录或其他机器上使用，方便管理和维护。SQLite 的默认编码是 UTF-8。

SQLite 和其他数据库最大的不同就是对数据类型的支持，创建一个表时，可以在 CREATE TABLE 语句中指定某列的数据类型，也可以把任何数据类型放入任何列中。当某个值插入数据库时，SQLite 将检查它的类型。如果该类型与关联的列不匹配，则 SQLite 会尝试将该值转换成该列的类型。如果不能转换，则该值将作为其本身具有的类型存储。例如，可以把一个字符串（String）放入 INTEGER 数据类型的列。这种特性称为“弱类型”。

SQLite 将数据值的存储划分为以下 5 种类型。

（1）NULL：表示该值为 NULL 值。

（2）INTEGER：带符号整型值。

（3）REAL：浮点值。

（4）TEXT：文本字符串，存储使用的编码方式为 UTF-8、UTF-16BE 和 UTF-16LE。

（5）BLOB：二进制对象，该类型数据和输入数据完全相同。

由于 SQLite 采用的是动态数据类型，而其他传统的关系型数据库使用的是静态数据类型，即字段可以存储的数据类型是在创建数据表时必须确定的，因此它们之间在数据存储方面还是存在着较大的差异。在 SQLite 中，存储分类和数据类型也有一定的差别，如 INTEGER 存储类别可以包含 6 种不同长度的整型数据类型，然而这些 INTEGER 数据一旦被读入到内存后，SQLite 会将其全部视为占用 8 个字节的整型。因此对于 SQLite 而言，即使在数据表中定义了明确的字段类型，仍然可以在该字段中存储其他类型的数据。然而需要特别说明的是，尽管 SQLite 为人们提供了这种方便，但是考虑到数据库平

台的可移植性问题，在实际的开发中还是应该尽可能地保证数据类型的存储和声明的一致性。

另外，SQLite 没有提供专门的布尔存储类型，取而代之的是整型 1 表示 true，0 表示 false。

SQLite 也同样没有提供专门的日期时间存储类型，而是以 TEXT、REAL 和 INTEGER 类型分别不同的格式表示该类型。TEXT 类型采用“YYYY-MM-DD HH：MM：SS. SSS”格式存储日期时间；REAL 类型以 Julian 日期格式存储，即自格林尼治时间公元前 4714 年 11 月 24 日中午以来的天数；INTEGER 类型以 UNIX 时间形式保存数据值，即从 1970-01-01 00：00：00 到当前时间的秒数。

8.2.2 创建数据库和表

Android 不自动提供数据库。在 Android 应用程序中如果使用 SQLite，就要创建数据库，然后创建表、索引等，然后就可以操纵数据了。

为了方便使用 SQLite 数据库，Android 提供了一些 API 类，主要有 SQLiteOpenHelper 类和 Cursor 类。SQLiteOpenHelper 类主要用于操作数据表中的数据，如建立、增、删、改、查等。Cursor 类主要用于遍历检索结果，处理从数据库查询出来的结果集。

SQLiteOpenHelper 类是 SQLiteDatabase 类的一个辅助类，是对数据库创建、版本更新等的管理类。此类是一个抽象类，使用时需要继承此类并实现该类的方法。只要继承 SQLiteOpenHelper 类，就可以创建数据库。

【例 8-4】 示例工程 08_DBCreateExample 演示了创建数据库的步骤。

创建数据库的步骤如下。

步骤 1： 创建工程项目 08_DBCreateExample。

步骤 2： 新建一个类 MyDBOpenHelper，继承自 SQLiteOpenHelper。

步骤 3： 重写 MyDBOpenHelper 的 3 个方法：构造方法、onCreate()方法和 onUpgrate()方法。

1. 构造方法

SQLiteOpenHelper 类要求必须重写其构造方法。构造方法有多重重载形式，重写其中一个即可。通常重写时会调用父类的下述构造方法创建一个数据库。

方法定义：

```
SQLiteOpenHelper(Context context,String name,CursorFactory factory,int version)
```

该构造方法需要 4 个参数：上下文环境（如 Activity）、数据库名字、可选的游标工厂（通常是 null）、代表正在使用的数据库模型版本的整数。

2. onCreate()方法

方法定义：

```
public void onCreate(SQLiteDatabase db)
```



onCreate()方法需要一个 SQLiteDatabase 对象作为参数，根据需要在数据库 db 中创建数据表和初始化数据。数据库第一次创建的时候会调用这个方法，一般在这个方法里边创建数据库表，写入初始化的数据。

3. onUpgrade()方法

方法定义：

```
public void onUpgrade(SQLiteDatabase,int,int)
```

onUpgrage()方法需要 3 个参数：SQLiteDatabase 对象、旧的版本号和一个新的版本号。当需要修改数据库定义的时候调用这个方法。一般在这个方法里修改数据库的结构。

当数据库需要升级的时候，Android 系统会主动地调用 onUpgrage()方法。一般在这个方法里删除数据库表，并建立新的数据库表。当然是否还需要做其他的操作，完全取决于应用程序的需求。

示例程序的主要代码如下：

```
//package 和 import 语句略
public class MyDBOpenHelper extends SQLiteOpenHelper {

    public MyDBOpenHelper(Context context) {
        //重写构造方法,在这里创建一个名为 DB_ContactList 的数据库
        super(context, "DB_ContactList", null, 1);
        }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //重写 onCreate()方法,创建数据表,其中 ID 字段作为主键
        String sql="create table tb_phones(id integer primary key autoincrement, name text, phone
        text, remark text);";
        db.execSQL(sql);                        //执行 SQL 语句
    }
    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        //TODO Auto-generated method stub
    }
}
```

本例中不需要升级数据库，所以在 onUpgrade()方法中没有执行任何操作。

在 Activity 中实例化这个类，就可以创建相应的数据库了。核心代码如下：

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.activity_main);
            MyDBOpenHelper dbHelper= new MyDBOpenHelper(this);
                //实例化 SQLiteOpenHelper 的子类,创建相应的数据库和数据表
        }
    }
```

与文件存取方式类似,SQLite 数据库文件存储在/data/data/＜package name＞/databases 目录中,如图 8-5 所示。默认状态下,该数据库文件只能由创建它的应用程序使用。其他 Activity 可以通过 Content Provider 或者 Service 访问这个数据库。

图 8-5　创建的数据库文件

继续调用 SQLiteDatabase 的 execSQL()方法来执行相应的语句可以完成创建表和索引的过程,如果没有异常,这个方法没有返回值。例如,示例工程的程序中执行如下语句:

```
String sql= "create table tb_phones(id integer primary key autoincrement, name text, phone text, remark
text);";
db.execSQL(sql);
```

上述语句创建了一个名为 tb_phones 的表,表有一个列名为 id 并且是主键,这列值的数据类型是整数,自动增值。SQLite 会自动为主键列创建索引。

这条语句在第一次创建数据库时创建了表和索引。如果不需要改变表的模式,是不需要删除表和索引的。如果要删除表和索引,可以调用 execSQL()方法执行 Drop Index 和 Drop Table 语句。

8.2.3 SQLite 数据库的查询操作

创建了数据库之后,可以使用 SQLiteOpenHelper 类的 getReadableDatabase()或 getWritableDatabase()方法得到 SQLiteDatabase 实例对象。SQLiteDatabase 封装了操纵数据库的各种方法,包括插入、删除、修改、查询、执行 SQL 命令等操作。获得了 SQLiteDatabase 对象以后,就可以通过调用 SQLiteDatabase 的实例方法来对数据库进行操作。当完成了对数据库的操作,需要调用 SQLiteDatabase 的 Close()方法来关闭数据库。

调用 getReadableDatabase()方法可以得到 SQLiteDatabase 实例对象,对数据库具

有读的权限。得到 SQLiteDatabase 实例对象后，有两种方法实现数据库的查询。

方法一：调用 SQLiteDatabase 对象的 rawQuery()方法，可以执行一条 SELECT 语句，实现数据库的查询。该方法的返回值是一个 Cursor 对象。

在 Android 系统中，数据库查询结果的返回值并不是数据集合的完整副本，而是返回数据集的指针，这个指针就是 Cursor。Cursor 类支持在查询的数据集合中以多种方式移动，并能够获取数据集合的属性名称和序号，并用于对查询结果进行操作等，可以对从数据库查询出来的结果集进行随机读写访问。执行 rawQuery()方法返回的 Cursor 对象最开始指向的是记录集合中第一行的上一行，因此首先需要先调用 moveToNext()方法将游标移动到记录集合的第一行，接着再获取数据即可。

Cursor 类提供了遍历数据库表的方法，其中常用方法如表 8-1 所示。

表 8-1　Cursor 类的常用方法

方　法　名	功　　能
boolean moveToPosition(position)	将游标移动到某记录
boolean moveToNext()	游标移动到下一条记录
getColumnNames()	得到字段名
getColumnIndex()	按列名获取 ID
int getCount()	获取记录总数
boolean isAfterLast()	游标是否在末尾
boolean isBeforeFirst()	游标是否在开始位置
boolean isFirst()	游标是否是第一条记录
boolean isLast()	游标是否是最后一条记录
boolean moveToFirst()/moveToLast()	游标移动到开始/末尾位置

【例 8-5】　工程 08_DBReadExample 演示了如何浏览数据库中的数据。

本例采用例 8-4 创建的数据库。界面布局中包括一些 TextView 控件和 EditText 控件，分别用于显示每一列数据的提示信息和数据内容。

MainActivity 类的主要代码如下：

```
public class MainActivity extends Activity {
    private Button BtnPre,BtnNext;
    private EditText TxtId,TxtName, TxtPhone, TxtRemark;
    private Cursor result;
    private SQLiteDatabase dbRead;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TxtId= (EditText)findViewById(R.id.Txt_id);
```

```
        TxtName= (EditText)findViewById(R.id.Txt_name);
        TxtPhone= (EditText)findViewById(R.id.Txt_phone);
        TxtRemark= (EditText)findViewById(R.id.Txt_remark);
        BtnPre= (Button)findViewById(R.id.btn_pre);
        BtnNext= (Button)findViewById(R.id.btn_next);
        MyDBOpenHelper dbHelper= new MyDBOpenHelper(this);
        dbRead= dbHelper.getReadableDatabase();
                //得到 SQLiteDatabase 实例对象,用于读数据
        refresh();
                //MainActivity类中自定义的 refresh()方法,
                //用于执行 SQL 查询语句,返回 Cursor 对象,读第一条数据

        BtnPre.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                readPre();
            }
        });
        BtnNext.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                readNext();
            }
        });
    }

    private void readPre() {
        result.moveToPrevious();
                //Cursor 指针向前移,用于读上一条数据
        if(!result.isBeforeFirst())    {
            TxtId.setText(String.valueOf(result.getInt(result.getColumnIndex("id"))));
            TxtName.setText(result.getString(result.getColumnIndex("name")));
            TxtPhone.setText(result.getString(result.getColumnIndex("phone")));
            TxtRemark.setText(result.getString(result.getColumnIndex("remark")));
          }
        else
        {
            Toast.makeText(this,"已经是第一条记录!", Toast.LENGTH_SHORT).show();
            result.moveToNext();
        }
    }

    private void readNext()
    {
```



```
            result.moveToNext();
                    //Cursor 指针向后移,用于读下一条数据
            if(!result.isAfterLast())
            {
                TxtId.setText(String.valueOf(result.getInt(result.getColumnIndex("id"))));
                TxtName.setText(result.getString(result.getColumnIndex("name")));
                TxtPhone.setText(result.getString(result.getColumnIndex("phone")));
                TxtRemark.setText(result.getString(result.getColumnIndex("remark")));
              }
            else
            {
                Toast.makeText(this,"已经是最后一条记录!", Toast.LENGTH_SHORT).show();
                result.moveToPrevious();
            }
        }

        private void refresh()
        {
            result=dbRead.rawQuery("select id, name, phone, remark from tb_phones", null);
                    //调用 rawQuery()方法,执行 SQL 语句,返回 Cursor 对象
            int resultCounts=result.getCount();
            if(resultCounts==0 || !result.moveToFirst()) {
                Toast.makeText(this,"数据库中无数据!", Toast.LENGTH_SHORT).show();
            }
            else {
                TxtId.setText(String.valueOf(result.getInt(result.getColumnIndex("id"))));
                TxtName.setText(result.getString(result.getColumnIndex("name")));
                TxtPhone.setText(result.getString(result.getColumnIndex("phone")));
                TxtRemark.setText(result.getString(result.getColumnIndex("remark")));
                    //获取第一行数据
            }
        }
    }
```

界面中一次显示一条数据，单击“上一个”按钮，则提取并显示上一条数据；单击“下一个”按钮，则提取并显示下一条数据，到达第一条数据或最后一条数据时，弹出一个 Toast 提示，数据指针不再移动。运行结果如图 8-6 所示。



图 8-6　浏览数据库中的数据

方法二：调用 SQLiteDatabase 对象的 query()方法查询数据库。

query()方法返回一个 Cursor 对象，方法的定义如下：

```
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy,
String having, String orderBy, String limit);
```

query()方法将 Select 语句的内容定义为其参数，除了表名，其他参数都可以是 null。query()方法中的参数意义如下。

(1) table：表名，不可为 null。

(2) columns：要返回的列名数组，如果其值为 null 则返回所有列。

(3) selection：相当于 SQL 语句的 where 子句，格式形如"_id=?"，此处将要填的参数写为"?"供下一个参数 selectionArgs 填充，如果其值为 null 则返回所有的行。

(4) selectionArgs：where 子句所需的值，该数组依次填充 selection 参数中的每一个问号。

(5) groupBy：定义查询是否分组，相当于 SQL 语句中的 group by 子句，如果其值为 null，则不分组。

(6) having：相当于 SQL 语句中的 having 短语，和 group 子句配套使用，表示对分组的筛选条件，如果 having 的值为 null，则保留所有的分组。

(7) orderBy：相当于 SQL 语句中的 order by 子句，描述对查询结果的排序要求，如果 orderBy 值为 null，将会使用默认的排序规则。

(8) limit 是可选的子句，如果其值为 null，将不会包含 limit 子句。

执行 query()方法后，与调用 SQLiteDatabase 对象的 rawQuery()方法执行一条 select 语句的结果相同，返回的也是一个 Cursor 游标，游标最开始指向的是记录集合中第一行的上一行。

例如：

```
SQLiteDatabase db=dbHelper.getReadableDatabase();
                        //获得拥有查询权限 SQLiteDatabase 对象
Cursor cur=db.query("mytable", null, null, null, null, null, null);
                        //指定查询 mytable 表,返回所有行和所有列
```

8.2.4 SQLite 数据库的更新操作

调用 getWritableDatabase()方法可以得到 SQLiteDatabase 实例对象，对数据库具有读写的权限。使用这个对象，调用其 execSQL()方法可执行一条更新数据的 SQL 语句，实现数据的插入、删除和修改。

SQLiteDatabase 的 execSQL()方法适用于所有不返回结果的 SQL 语句，当 SQL 语句为 INSERT、UPDATE、DELETE 时，可采用调用 execSQL()方法的方式执行。

例如，向数据库的表中插入一行记录('李庆华','13933105035','北京')，可通过执行以下语句实现：

```
SQLiteDatabase db=dbHelper.getWritableDatabase();
            //获取拥有修改权限的 SQLiteDatabase 实例对象
db.execSQL("insert into tb_phones(name, phone, remark)values('李庆华','13933105035','北京')");
```

将数据库的表中的记录('李庆华','13933105035','北京')修改为('李庆华','13933105035','上海')，假设数据库中只有一条姓名为"李庆华"的记录，则可通过执行以下

语句实现。

```
SQLiteDatabase db=dbHelper.getWritableDatabase();
        //获取拥有修改权限的 SQLiteDatabase 实例对象
db.execSQL("update tb_phones set remark='上海' where name='李庆华';");
```

将数据库的表中的记录('李庆华','13933105035','北京')删除，假设数据库中只有一条姓名为“李庆华”的记录，可通过执行以下语句实现：

```
SQLiteDatabase db=dbHelper.getWritableDatabase();
        //获取拥有修改权限的 SQLiteDatabase 实例对象
db.execSQL("delete from tb_phones where name='李庆华';");
```

【例 8-6】 工程 08_DBInsertExample 演示了向数据库中插入数据。

Activity 的界面中设置了 3 个 EditText 控件，分别用于输入 3 个字段的内容，单击“添加”按钮，执行 insert 语句完成数据插入。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private Button BtnInsert;
    private EditText TxtName, TxtPhone, TxtRemark;
    private SQLiteDatabase dbWrite;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TxtName=(EditText)findViewById(R.id.Txt_name);
        TxtPhone=(EditText)findViewById(R.id.Txt_phone);
        TxtRemark=(EditText)findViewById(R.id.Txt_remark);
        MyDBOpenHelper dbHelper=new MyDBOpenHelper(this);
        dbWrite=dbHelper.getWritableDatabase();
        BtnInsert=(Button)findViewById(R.id.btn_insert);
        BtnInsert.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                String sql="insert into tb_phones(name, phone, remark)values('"+ TxtName.getText()
                +"','"+TxtPhone.getText()+"','"+TxtRemark.getText()+"');";
                dbWrite.execSQL(sql);
                Toast.makeText(MainActivity.this,"数据添加成功!", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

单击“添加”按钮后，运行结果如图 8-7 所示。

与数据库查询的方法类似，除了调用 execSQL()方法执行 SQL 语句可以实现数据更新

以外，还可以直接调用 SQLiteDatabase 对象的 insert()、update()和 delete()方法，分别实现数据的插入、修改和删除。这些方法把 SQL 语句的一部分作为参数。



图 8-7 向数据库中插入数据

1. 插入数据

向数据库的表中插入记录时，需要先将数据包含在一个 ContentValues 对象中。ContentValues 类是一个数据承载容器，主要用来向数据库表中添加一条数据。使用方法是创建一个 ContentValues 对象，并调用 put()方法向该对象当中插入键-值对，其中键是列名，值是希望插入到这一列的值。

insert()方法的定义如下：

```
long insert(String table, String nullColumnHack, ContentValues values)
```

其中，第 1 个参数是想要插入数据的表名；第 2 个参数 nullColumnHack 用于指定空值字段的名称，当初始化值为空行时，这一列将会被显式地赋一个 null 值；第 3 个参数 values 是要插入的值。

insert()方法的第 2 个参数用于指定空值字段的名称，当 values 参数值为 null 或者元素个数为 0 的时候，因为数据库不允许插入一个空行，插入操作会失败。为了防止这种情况，要在这里指定一个列名，如果将要插入的行为空行时，系统会将指定的这个列名的值设为 null，然后再向数据库中插入。

insert()方法的返回值是新添记录的行号，与主键 id 的值无关。

以下是实现插入数据的一段示例代码：

```
ContentValues cv=new ContentValues();
cv.put("name", "李庆华");                      //将值存放到对应的键中
cv.put("phone", "13933105035");
cv.put("remark", "北京");
dbWrite.insert("tb_phones", "name", cv);      //插入,返回新添记录的行号
```

2. 修改数据

调用 update()方法，可以修改表中的数据。update()方法根据条件更新指定列的值，返回 int 值。update ()方法定义如下：

```
int update(String table, ContentValues values, String whereClause, String[] whereArgs)
```

update()方法有 4 个参数：table 是想要修改数据的表名；values 是要更新的值；whereClause 是可选的参数，如果其值为 null，将会修改所有的行；whereArgs 是一个字符串数组，当在 whereClause 中包含“?”时，这个数组中的值将依次替换 whereClause 中出现的“?”。

以下是实现将数据库的表中的记录(11,'李庆华','女')修改为(11,'李庆华','男')的一段示例代码,其中 11 是主键值。

```
String where="authorID=?";
String[] whereValue={ Integer.toString(11)};
ContentValues cv=new ContentValues();
cv.put("sex", "男");                              //键-值对
db.update("tb_author", cv, where, whereValue);
```

3. 删除数据

调用 delete()方法,从表中删除一行数据。delete()方法定义如下:

```
int delete(String table, String whereClause, String[] whereArgs)
```

其中,参数 table 是想要删除数据的表名;参数 whereClause 是可选的 where 子句,如果其值为 null,将会删除所有的行;参数 whereArgs 是一个字符串数组,当在 whereClause 参数中包含"?"时,这个数组中的值将依次替换 whereClause 中出现的"?"。

以下是实现将数据库的表中的记录(11,'李庆华','男')删除的一段示例代码,其中 11 是主键值。

```
String where="authorID=?";
String[] whereValue={Integer.toString(11)};
db.delete("tb_author", where, whereValue);
```

8.2.5 使用 sqlite3 工具管理数据库

Android SDK 的 tools 目录下提供了一个 sqlite3.exe 工具,用于 SQLite 数据库的操纵和管理。可以对 SQLite 数据库进行命令行的操作。

一般数据库文件以.db 结尾,位于/data/data/<package name>/databases/目录里,要对数据库文件进行操作需要先找到数据库文件。

以下操作都在命令行模式下进行,进入命令行模式的方法:在 Windows 中执行"开始"→"附件"→"命令提示符"命令,打开命令提示符窗口。在窗口中使用 cd 命令进入到"\SDK 安装路径\tools"目录。

1. 打开或创建数据库

在命令行下输入命令"sqlite3 数据库名",就可以打开指定的数据库,并且进入到 sqlite>提示符下。如果指定的数据库不存在,则自动创建一个新的数据库文件。例如:

```
D:\SDK安装路径\tools>sqlite3  testdatabase.db
```

这样就能打开或者创建数据库文件 testdatabase.db。命令执行后系统会出现 sqlite>提示符,如图 8-8 所示。

在 sqlite>提示符下可以直接输入以";"结尾的 SQL 命令,也可以输入以"."开始的

```
G:\adt-bundle-windows-x86-20140321\sdk\tools>sqlite3 testdatabase.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

图 8-8　运行 sqlite3 命令后的系统提示

SQLite 内置命令。常用的内置命令如表 8-2 所示。

表 8-2　常用的 SQLite 内置命令

SQLite 命令	说　明
. help	查看 sqlite3 帮助
. databases	查看当前数据库
. tables	查看当前数据库中的表,查看当前有多少表
. schema	查看各个表的生成语句
. headers ON	查询结果列出表头
. backup d:/mydb. db	将当前连接中的缓存数据导出到本地文件
. restore d:/mydb. db	将导出的数据库作为主库重新导入
. read ××. sql	导入 SQL 批处理文件,并执行
. quit	退出 sqlite3
. exit	退出 sqlite3

2. 创建表

在 sqlite>提示符下直接输入创建表的 SQL 语句,就可以实现创建新表的操作。例如:

```
sqlite> create table tb_author(authorID integer PRIMARY KEY, authorName text, sex text);
sqlite> create table tb_book(bookID integer PRIMARY KEY, bookName text, authorID integer, publishTime text);
```

这里需要注意的是,对于自定义数据表表名,不能以 sqlite_开头,因为该前缀定义的表名都用于 SQLite 内部。

SQLite 对 SQL 语句的大小写不敏感。SQL 语句以分号结束,一条语句可以多行书写,直到输入分号,系统才会执行这条 SQL 语句。如果 SQL 命令没有以分号结束时按 Enter 键,系统会将提示符改为“...>”,如图 8-9 所示。

```
sqlite> CREATE TABLE tb_book(
   ...> bookID integer PRIMARY KEY,
   ...> bookName text,
   ...> publishtime text);
```

图 8-9　多行输入 SQL 语句

3. 表的删除

在 sqlite>提示符下直接输入删除表的 SQL 语句,就可以实现表的删除操作。在 SQLite 中如果被删除的表不存在,SQLite 将会报错并输出错误信息。如果希望在执行



时不抛出异常，可以在 SQL 语句中添加 IF EXISTS 短语。例如：

```
sqlite> drop table if exists testtable;
```

4. 插入、修改和删除数据

在 sqlite>提示符下直接输入插入、修改和删除数据的 SQL 语句，就可以实现对数据库的数据操纵。例如，下面的命令实现数据插入：

```
sqlite> insert into tb_author(authorID,authorName,sex)values(3,'赵云坤','男');
```

需要注意的是，在 SQLite 中，null 值被视为和其他任何值（包括其他的 null 值）都是不同的，如下例，虽然 authorID 是主键，但两次插入的 null 值均插入成功。

```
sqlite> insert into tb_author(authorID,authorName,sex)values(null,'李华','女');
sqlite> insert into tb_author(authorID,authorName,sex)values(null,'刘书','男');
```

5. 查询数据

在 sqlite>提示符下直接输入查询的 SQL 语句，就可以实现对数据库的数据查询。可以使用带条件的查询语句，也可以对多表进行连接查询。例如，下面的命令返回 tb_author 表中的所有数据：

```
sqlite> select * from tb_author;
```



图 8-10　查询语句的执行结果

输出结果如图 8-10 所示。

命令行操纵数据库的方法虽然简单，但不够直观高效。除了使用 sqlite3 命令行工具以外，还可以使用第三方的 SQLite 可视化管理工具来管理 SQLite 数据库，例如 SQLiteSpy、SQLiteStudio、SQLiteExpert 和 SQLite Manager 等。它们一般具有图形用户界面，操作过程更为直观。

8.2.6　基于 SQLite 数据库的综合应用示例

【例 8-7】 工程 08_DBPhonesListExample 演示了对 SQLite 数据库的各种操作。应用程序的主要功能是将姓名、电话号码等存储到数据库中，并能够增加新电话号码、对数据库中的数据单条或列表显示。

示例工程的主要实现步骤如下。

步骤 1：创建一个工程，工程名为 08_DBPhonesListExample，并填写相应的工程信息。

步骤 2：工程创建完成之后，根据工程所要实现的功能建立用户界面，在这个应用程序中有 3 个 Activity 界面，分别是单条显示的 MainActivity 界面，如图 8-11 所示；添加新

电话号码的 InsertActivity 界面，如图 8-12 所示；列表显示的 ListActivity 界面，如图 8-13 所示。

图 8-11 单条显示

图 8-12 添加新电话

图 8-13 列表显示

这 3 个 Activity 分别使用布局文件 main. xml、insert. xml 以及 list. xml 和 listitem. xml。

步骤 3：在这个示例中定义了一个数据表，表名为 tb_phones，该表包含 4 个属性列：id（唯一编号）、name（姓名）、phone（电话号码）和 remark（备注）。

通过定义继承自 SQLiteOpenHelper 的 DBHelper 类来创建数据库，主要代码如下：

```
//import 语句略
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(Context context) {
          //重写构造方法,在这里创建一个名为 DB_ContactList 的数据库
        super(context, "DB_ContactList", null, 1);
    }
    public void onCreate(SQLiteDatabase db) {
        //新建一个数据表,其中 ID字段作为主键
        String sql= "create table tb_phones(id integer primary key autoincrement, name text, phone
        text, remark text);";
        db.execSQL(sql);                          //执行 SQL语句
    }
```

本例中，重写了 SQLiteOpenHelper 类的 onCreate()方法创建数据表 tb_phones。在执行 SQLiteOpenHelper 类对象的 getWritableDatabase()或 getReadableDatabase()方法的时候，如果数据库不存在，onCreate()方法会自动被调用。

步骤 4：在 Activity 中操纵数据库。操纵数据库之前需要首先构造 DBHelper 对象，然后通过调用其 getWritableDatabase()或 getReadableDatabase()方法获得一个 SQLiteDatabase 对象。获得了 SQLiteDatabase 对象后，就可以对其进行增、删、改、查等操作，例如：



查询数据库并将结果显示到 ListView 控件中的主要代码如下：

```
String idStr,nameStr, phoneStr, remarkStr;
ArrayList<String> phons=new ArrayList<String> ();  //每个元素是一个人的信息
DBHelper dbHelper=new DBHelper(this);
SQLiteDatabase dbRead=dbHelper.getReadableDatabase();
                    //调用 getReadableDatabase()方法获得一个只读的 SQLiteDatabase 对象
Cursor result=dbRead.rawQuery("select id, name, phone, remark from tb_phones", null);
                    //调用其 rawQuery()方法执行 SQL 查询语句
int resultCounts=result.getCount();
if(resultCounts==0 || !result.moveToFirst()) {
     Toast.makeText(this,"数据库中无数据!", Toast.LENGTH_SHORT).show();
  }
else  {
     while(!result.isAfterLast()) {
          dStr=String.valueOf(result.getInt(result.getColumnIndex("id")))+"#\t";
          nameStr=result.getString(result.getColumnIndex("name"))+" : \t";
          phoneStr=result.getString(result.getColumnIndex("phone"))+" :\t";
          remarkStr=result.getString(result.getColumnIndex("remark"));
          phons.add(idStr+nameStr+phoneStr+remarkStr);
          result.moveToNext();
        }
  }
ListView ListPhone=new ListView(this);
ListPhone.setAdapter(new ArrayAdapter<String> (this,R.layout.listitem,phons));
                    //将查询到的结果显示到 ListView 控件中
```

示例程序在实现插入数据时，采用的方法是调用 SQLiteDatabase 对象的 insert()方法，核心代码如下：

```
public void onClick(View v) {
     switch(v.getId()) {
        case R.id.btn_insertNew:
          if(TxtName.getText().toString().length()!=0) {
               nameStr=TxtName.getText().toString();
           }
          else {
              Toast.makeText(getApplication(), "姓名不能为空", Toast.LENGTH_SHORT).show();
              return;
           }
          if(TxtPhone.getText().toString().length()!=0) {
              phoneStr=TxtPhone.getText().toString();
          }
          else {
              Toast.makeText(getApplication(), "电话号码不能为空", Toast.LENGTH_SHORT).show();
```

```
                return;
            }
            ContentValues values=new ContentValues();
            remarkStr=TxtRemark.getText().toString();
            values.put("name", nameStr);
            values.put("phone",phoneStr);
            values.put("remark",remarkStr);
                    //实例化 ContentValues 对象,填充数据,键是数据库的列名,值是要插入的列值
            dbWrite.insert("tb_phones", "name", values);
                    //调用 insert()方法插入数据库
            TxtName.setText("");
            TxtPhone.setText("");
            TxtRemark.setText("");
            Toast.makeText(getApplication(), "数据添加成功!", Toast.LENGTH_SHORT).show();
            break;
        }
}
```

步骤 5：在 AndroidManifest. xml 配置文件中注册所有 Activity。

```
<activity
    android:name="edu.hebust.zxm.dbphoneslist.MainActivity"
    android:label="我的通讯录">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name="edu.hebust.zxm.dbphoneslist.InsertActivity"
    android:label="添加新电话">
</activity>
<activity
    android:name="edu.hebust.zxm.dbphoneslist.ListActivity"
    android:label="通讯录电话列表">
</activity>
```

在此示例程序中，数据库文件 DB_PhonesList 存放在工程的/data/data/edu. hebust. zxm. dbphoneslist/databases 目录中，如图 8-14 所示。

通过 DDMS 窗口右上方的 pull a file from the device 按钮，可以将这个文件导出到 PC 中指定的目录下。但这个数据库文件是不能直接打开的。可以使用 8. 2. 5 节介绍的方法，利用 sqlite3 工具支持的 SQL 命令查询数据库中的内容，也可以安装一个可视化 SQLite 数据库管理工具，打开数据库文件。如图 8-15 所示为 SQLiteSpy 的运行界面，在左侧列表中列出了数据库中的各表及其结构，选择其中一个就可在右边的选项卡中看到



表中的数据。

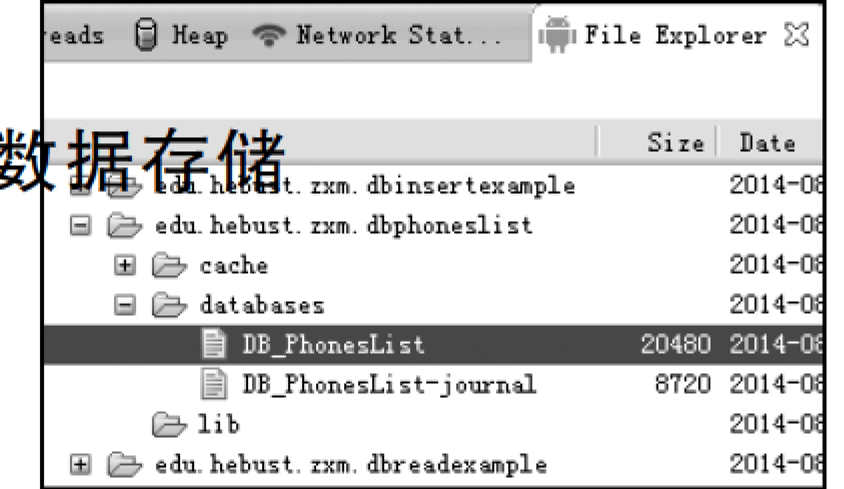


图 8-14　数据库文件

8.3　利用内容提供器 ContentProvider 共享数据存储

在 Android 中，通过文件和 SQLite 数据库可以存储数据，但是这些数据都是应用程序私有的，如果多个应用程序需要共享同样的数据，那么就需要使用 ContentProvider。

8.3.1　自定义 ContentProvider

ContentProvider 是 Android 的主要组件之



图 8-15　SQLiteSpy 显示数据库表中的数据

一，它提供了应用程序间共享数据的机制和数据存储方式。系统为所有的 ContentProvider 建立了一个数据表 Data Model，Data Model 中保存了系统和用户共享的所有数据集，每个数据集就像数据库中的数据表一样，用_ID 区别每条数据，每一列代表每条数据的属性。每个 ContentProvider 对象都对外提供了一个公开的 Uri 来表示相应的数据集。

如果应用程序有数据需要共享时，就可以使用 ContentProvider 为这些数据定义一个 Uri，其他的应用程序就可以通过这个 Uri 来对数据进行操作。

ContentProvider 使用的 Uri 通常有两种形式，一种是指定全部数据，例如，content://PhonesList/phones 指的是全部通讯录数据；另一种是某个指定 ID 的数据。例如，content://PhonesList/phones/1 指的是 id 列值为 1 的通讯录数据。

所有的 Uri 均由三部分组成：scheme、authority/host 和 path，它们的含义如下。

（1）scheme：对于 ContentProvider，Android 规定的 scheme 为“content://”。

（2）authority/host：授权者或主机名称，用于唯一标识一个 ContentProvider。外部应用程序可以根据这个标识来找到相应的共享数据。一般 authority 都由类的小写全称组成，以保证唯一性。

(3) path：要操作的数据路径，用于确定请求的是哪一个数据集。

ContentProvider 与 Service、BroadcastReceiver 等组件一样，在 AndroidManifest.xml 文件里声明后调用者就可以使用了。

自定义的 ContentProvider 和系统 ContentProvider 在使用上并没有什么区别，只不过系统 ContentProvider 在使用时需要注册其使用权限并知道它的 Uri，而自定义的则不需要，只需要知道它的 Uri 即可，下面是自定义 ContentProvider 的主要步骤。

(1) 创建自己的数据存储，如数据库、文件或其他。

(2) 创建一个继承于 ContentProvider 类的子类。在子类中重写 ContentProvider 的 6 个抽象方法 query()、insert()、update()、delete()、getType()和 onCreate()，定义查询、插入、修改、删除和创建等操作接口。

(3) 在 AndroidManifest.xml 文件中注册新定义的 ContentProvider 及其对外共享标识 Uri。

定义完成后，在其他应用程序中就可以对共享的 ContentProvider 数据进行操作。

8.3.2 使用 ContentProvider 共享数据

应用程序可以通过 ContentResolver 接口存储和读取 ContentProvider 共享数据。在 Activity 中，可以通过调用 getContentResolver() 方法得到当前应用的 ContentResolver 实例对象。ContentResolver 提供的接口和 ContentProvider 中需要实现的抽象方法对应，主要有 query()、insert()、update()和 delete()等，分别通过 Uri 进行查询、插入、修改和删除。

【例 8-8】 工程 08_DBContentProviderExample 将示例工程 08_DBPhonesList 进行了修改，将数据库中的信息存储在 ContentProvider 中，这样做的好处是如果以后另外一个程序需要访问此数据库中的数据时，只需要知道 ContentProvider 的 Uri 即可。

示例工程的实现过程。

步骤 1：定义一个继承 SQLiteOpenHelper 类的 DBOpenHelper 类，创建数据库和数据表，创建的数据库要将其共享到 ContentProvider 中。

步骤 2：创建 ContentProvider 的子类 MyDBProvider。在创建该类时，必须要重写 ContentProvider 类的 6 个抽象方法 query()、insert()、update()、delete()、getType()和 onCreate()。其中前 4 个抽象方法分别对应于 ContentResolver 的 query()、insert()、update()、delete()，当调用 ContentResolver 的这 4 个方法时，也就间接调用了 ContentProvider 的 4 个方法。这 4 个方法重写的代码如下：

```
private DBOpenHelper dbOpenHelper;
@Override
public boolean onCreate() {
    dbOpenHelper=new DBOpenHelper(getContext());    //实例化 DBOpenHelper 对象
    return true;
}
```




```
//获得数据库对象,并进行删除操作,返回删除的行数
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    SQLiteDatabase db=dbOpenHelper.getWritableDatabase();
    return db.delete("tb_phones", selection, selectionArgs);
}

//获得数据库对象,并进行添加操作,返回添加最新行的 Uri
@Override
public Uri insert(Uri uri, ContentValues values) {
    SQLiteDatabase db=dbOpenHelper.getWritableDatabase();
    long i=db.insert("tb_phones",null,values);
    uri=ContentUris.withAppendedId(uri, i);
    return uri;
}

//获得数据库对象,并进行查询操作,返回 Cursor 对象
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
    SQLiteDatabase db=dbOpenHelper.getWritableDatabase();
    Cursor c=db.query("tb_phones", projection, selection, selectionArgs, null, null,sortOrder);
    return c;
}

//获得数据库对象,并进行修改操作,返回修改的 row 值
@Override
public int update(Uri uri, ContentValues values, String selection,String[] selectionArgs) {
    SQLiteDatabase db=dbOpenHelper.getWritableDatabase();
    return db.update("tb_phones", values, selection, selectionArgs);
}
```

ContentProvider 的增、删、改、查等操作实际上是间接调用了数据库 db 的对应操作完成的。

步骤 3：完成了 ContentProvider 子类的创建及方法重写，就要把新生成的 ContentProvider 告诉给系统，此时需要在 AndroidManifest.xml 配置文件中注册自己的 ContentProvider：

```
<provider  android:name="MyDBProvider"
           android:authorities="edu.hebust.zxm.dbcontentprovider">
</provider>
```

代码中的 name 值 MyDBProvider 是程序中对应的 ContentProvider 的子类；authorities 的值就是这个 ContentProvider 对外公开的 Uri。其他应用程序使用这些数据

就是根据这个 Uri 来找到它。该 Uri 的格式是 content://edu. hebust. zxm. dbcontentprovider/tb_phones，其中 Content：指的是内容提供者 ContentProvider，//edu. hebust. zxm. dbcontentprovider 映射到人们已定义的那个 ContentProvider 标识，/tb_phones 作为一个参数传给 ContentProvider，可以根据这个参数来决定操作目标，例如数据库中的哪个表，文件中的哪一部分数据等。

步骤 4：注册了 ContentProvider 后，其他程序就可以对 ContentProvider 中的数据进行增、删、改、查等操作了。

本例中 MainActivity 的界面布局如图 8-16 所示，使用 Uri 的方式访问 ContentProvider 中的共享数据。

图 8-16　MainActivity 的界面布局

本例与工程 08_DBPhonesList 的不同之处仅仅在于访问数据库的方式，不是使用 SQLiteDatabase 对象，而是使用 ContentResolve 对象。查询数据库的代码片段如下：

```
String url="content://edu.hebust.zxm.dbcontentprovider/tb_phones";
private ArrayList<String> phons=new ArrayList<String>();
Cursor result=getContentResolver().query(Uri.parse(url),
        new String[]{"id","name","phone","remark"}, null, null, null);
if(result.getCount()==0) {
    Toast.makeText(this,"数据库无数据!", Toast.LENGTH_SHORT).show();
}
else {
    //略,与工程 09_Datebase_ConList 相同
    ListPhone.setAdapter(new ArrayAdapter<String>(this, R.layout.listitem, phons));
    //将查询到的结果显示到 ListView 控件中
}
```

代码中，通过调用 getContentResolver()方法得到 ContentResolve 对象，然后调用该对象的 query(Uri uri，String[] projection，String selection，String[] selectionArgs，String sortOrder)方法对 ContentProvider 进行查询。根据要查询数据的 ContentProvider 的 Uri 值找到共享的数据表。

8.3.3　系统 ContentProvider

Android 系统为常见的一些数据提供了 ContentProvider，包括音频、视频、图片和联系人等，每个 ContentProvider 都会对外提供一个包装成 Uri 对象的公共 URI。这些

ContentProvider 称为系统 ContentProvider。

可以利用系统 ContentProvider 存放一些相同格式的数据，但是在使用时必须要注册相应的系统权限。假如要读取手机联系人的数据，那么就必须在 AndroidManifest.xml 文件中注册相应的 READ_CONTACTS 使用权限，即在 AndroidManifest.xml 文件中加入下面的语句：

```
<uses-permission
    android:name="android.permission.READ_CONTACTS">
</uses-permission>
```

当然，不同的 ContentProvider 所要求注册的系统权限也是不相同的，系统 ContentProvider 和用户自定义的 ContentProvider 使用方法相同。首先，用 getContentResolver()方法得到 ContentResolver 对象，然后通过调用 ContentResolver 的 query()方法或 Activity 的 managedQuery()方法，来查询 ContentProvider 中的数据；调用 update()方法来修改数据，调用 insert()方法来添加数据；调用 delete()方法来删除数据。

【例 8-9】 工程 08_SystemProviderExample 演示了利用系统 ContentProvider 将联系人中的信息用列表显示出来。

首先在系统联系人应用中添加一些联系人信息。单击模拟器的 home 键，单击手机桌面的“联系人”应用，添加新联系人。添加完成后，在返回的界面中保存。

新建工程项目 08_SystemProviderExample。在 MainActivity 中重写 onCreate()方法。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView ListPhone= (ListView)findViewById(R.id.listphones);
        Cursor result= getContentResolver().query(Phones.CONTENT_URI, null, null, null, null);
        //得到 ContentResolver 实例,query()的功能是查询所有的联系人
        ListAdapter adapter= new SimpleCursorAdapter(this,R.layout.listitem,result,
                new String[]{Phones.NAME,Phones.NUMBER},
                new int[]{android.R.id.text1,android.R.id.text2});
        ListPhone.setAdapter(adapter);                    //将 ListView 和 SimpleCursorAdaper 绑定
    }
}
```

程序的运行结果如图 8-17 所示。

8.4　本 章 小 结

本章学习了在 Android 中如何实现数据的存储和获取，以及如何在应用程序之间利



图 8-17 示例程序的运行结果

用 ContentProvider 共享数据存储。利用文件和 SQLite 数据库都可以存储大容量的数据，它们具有不同的存储机制和操作方法。文件存取采用 java.io.* 库提供的 I/O 接口读写文件，SQLite 数据库利用 Android 提供的 SQLiteDatabase 类对其访问，该类封装了一些操作数据库的 API，可以完成对数据进行查询、插入、修改和删除操作。ContentProvider 是用于在不同应用程序之间共享数据的一个接口，每个 ContentProvider 都会对外提供一个公共的 Uri，其他的应用程序可以通过这个 Uri 来对数据进行操作。学习本章要求重点掌握文件存取的操作方法和 SQLite 数据库的操作方法。

习 题

1. Android 系统提供了哪些数据存储和访问方式？

2. 简述 SQLite 和 ContentProvider 的区别。

3. 简述自定义 ContentProvider 的步骤。

4. 简述 ContentProvider 是如何实现数据共享的，尝试设计一个属于自己的 ContentProvider。

5. 尝试将你的数据添加到一个已经存在的 ContentProvider 中，前提是有相同数据类型并且有写入 ContentProvider 的权限。

6. 编写一个应用程序，其界面如图 8-18 所示，要求用户输入文件名、文件内容，单击“保存”按钮，则按照指定的文件名保存输入的文字内容。

7. 继续完成第 5 章习题 6 设计的注册页面。要求单击“注册”按钮后，将所有提交的注册信息存储到一个数据文件中，每行存储一项注册信息，文件名为 count.txt。

8. 继续完成第 5 章习题 6 设计的注册页面。要求单击“注册”按钮后，将所有提交的注册信息存储到数据库的 users 表中。

9. 改写例 8-5(工程 08_DBReadExample)，调用 SQLiteDatabase 对象的 query()方法完成数据库的查询。

10. 设计一个利用 SQLite 数据库存储和操纵数据的程序，要求新建一个存储课程信息的数据表(course)，在课程信息编辑框中输入课程信息(不包括课程 ID 课程，ID 要自动生成)，单击“插入”按钮后，数据插入到 course 表中。

11. 改写例 8-7(工程 08_DBPhonesListExample)，定义一个实体类 Person.java，包



含通讯录的属性信息。类的属性定义如下：id（唯一编号）、name（姓名）、phone（电话号码）和 remark（备注）。实现程序的控制过程。

12．编写一个程序，继承 SQLiteOpenHelper 实现下述功能：创建一个版本为 1 的 diary.db 数据库，同时创建一个 diary 表，包含一个_id 主键并自增长，以及 topic 字段（字符型，最大长度 100 个字符）和 content 字段（字符型，最大长度 1000 个字符），在数据库版本变化时删除 diary 表，并重新创建出 diary 表。



图 8-18　习题 6 的程序界面

13．编写一个应用程序，创建一个商品基本信息表 product，包含商品编号、名称、价格和描述 4 个字段，实现表数据的增、删、改、查。

14．在 AndroidManifest.xml 配置文件中注册 ContentProvider 的目的是什么？如何进行注册？



第9章 图片和音视频的处理

在智能移动设备的应用中，照片、音视频等多媒体应用是一个重要的方面。本章将介绍在 Android 系统如何处理和使用图片、音视频等资源，包括提取和显示媒体库的图片、照片的摄取、音频和视频的播放及录制等。

9.1 相关控件和类

9.1.1 ImageView

ImageView 是 android. view. View 的子类，在 android. widget 包中。ImageView 控件用于显示各种来源的图像，包括资源文件中的 ID、Drawable 对象、位图对象或 ContentProvider 图片库等。ImageView 显示图像时可以对图像进行缩放等处理，利用其 scaleType 属性可以控制调整图像的大小和位置。

【例 9-1】 工程 09_ImageViewExample 演示了如何使用 ImageView 控件显示图像文件。

在布局文件 Activity_main. xml 中添加 ImageView 控件，本例中 scaleType 属性值设为 centerInside，图片设置为居中显示。src 属性用于设置图像文件的来源，本例为 drawable 文件夹中的 flower600. jpg 文件。Activity_main. xml 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"  >
  <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="使用 ImageView 控件显示图片:\n" />
  <ImageView
       android:id="@+id/Iv_image"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:src="@drawable/flower600"
```

```
            android:scaleType="centerInside" />
</LinearLayout>
```

在 MainActivity 中引用该布局，运行结果如图 9-1 所示。



图 9-1　使用 ImageView 控件显示图片

9.1.2　ImageButton

ImageButton 是 android. widget. ImageView 的子类，在 android. widget 包中。ImageButton 控件显示一个可以被用户单击的图片按钮，按钮的图片可以在 XML 文件中通过<ImageButton>标签的 android:src 属性或在 Java 代码中调用 setImageResource (int)方法指定。ImageButton 与 Button 的最大区别是没有 text 属性。

为了表示不同的按钮状态，如获得焦点、按下等，可以为各种状态定义不同的图片。此功能可以通过 XML 文件中的<selector>标签下的<item>元素配置，文件通常保存到 res/drawable/文件夹下。

保存上述文件后，将该文件名作为一个参数设置到 ImageButton 的 android:src 属性。例如，android:src="@drawable/myselector"，表明按钮的外观使用 res/drawable/myselector. xml 文件中的配置。Android 根据按钮的状态改变会自动地去 myselector. xml 中查找相应的图片显示。

【例 9-2】　工程 09_ImageButtonExample 演示了图片按钮的设置和使用方法。

首先在布局文件 Activity_main. xml 中添加 ImageButton 控件，外观使用 res/drawable/myselector. xml 文件中的配置，主要代码如下。

```
<ImageButton
    android:id="@+id/Iv_image"
    android:layout_width="wrap_content"
```

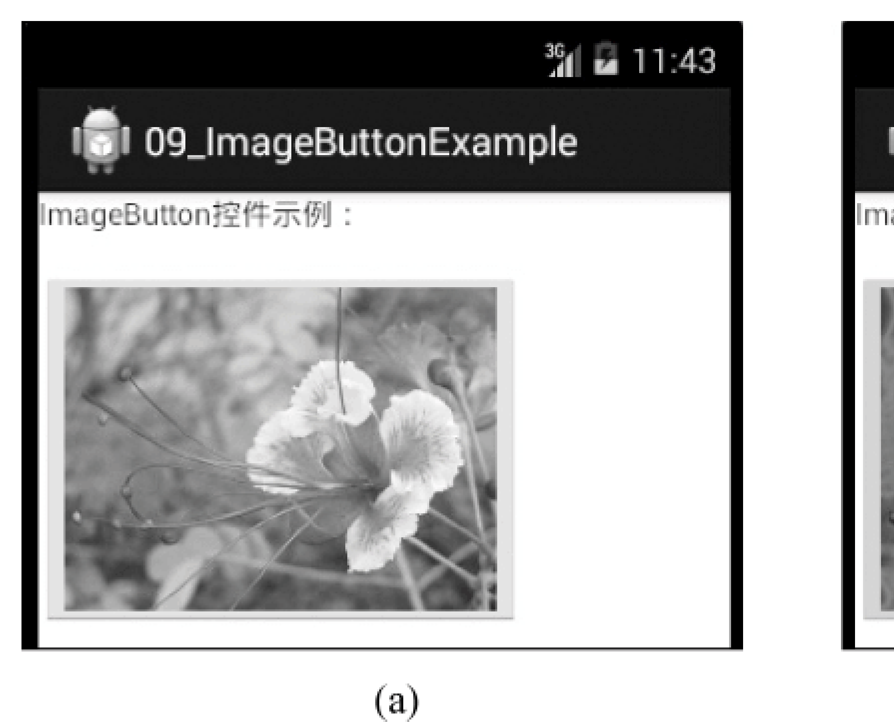
```
        android:layout_height="wrap_content"
        android:src="@drawable/myselector" />
```

在 myselector.xml 文件中设置按钮默认图片为 flower400.jpg，获取焦点时显示的图片为 flower200.jpg，按钮被按下时显示的图片为 flower300.jpg。这些图片文件放置在 res/drawable 文件夹中。myselector.xml 文件内容为如下：

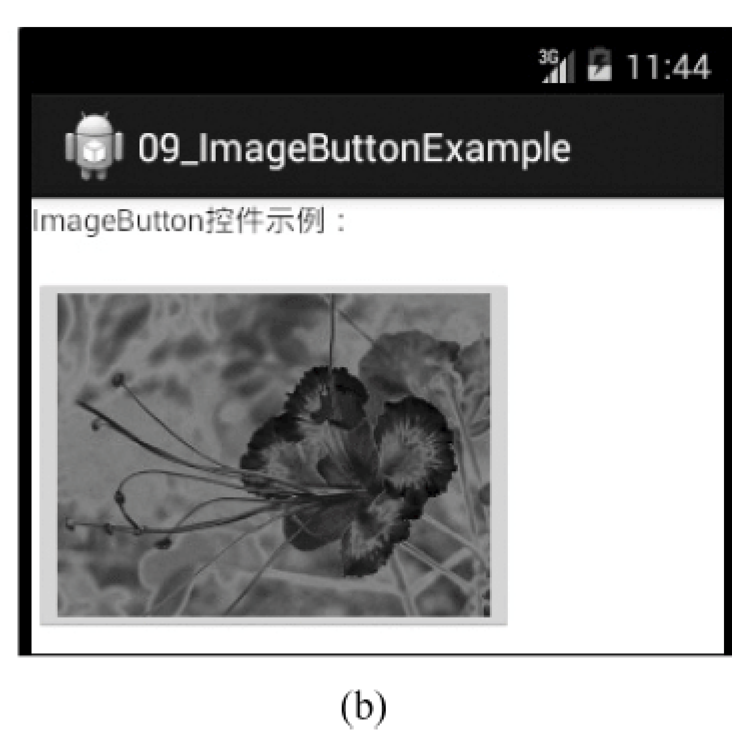
```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_focused="true"
        android:drawable="@drawable/flower200"></item>
    <item android:state_pressed="true"
        android:drawable="@drawable/flower300"></item>
    <item android:drawable="@drawable/flower400"></item>
</selector>
```

需要注意的是，<item>元素的顺序很重要。因为系统是根据这个顺序判断是否适用于当前按钮状态，一旦匹配到一个合适的状态，就不会继续寻找。例如，按钮被按下时会同时获得焦点，但是获得焦点并不一定按了按钮，出现这种情况系统会按顺序查找，找到合适的<item>元素就不再继续查找了。如果按钮被单击了，那么按钮被单击的<item>元素将被选中，且不再向后面查找其他状态。所以，一般按钮的正常或默认状态指定的图片放在最后，是因为它只会在 pressed 和 focused 都判断失败之后才会被采用。

示例程序的运行结果如图 9-2 所示，其中图 9-2(a)为按钮的正常状态，图 9-2(b)为按钮被按下时的状态。



(a)



(b)

图 9-2　ImageButton 按钮被按下之前和之后的效果

9.1.3　SurfaceView

SurfaceView 继承自 android.view.View 类，视图里内嵌了一个专门用于绘制的 Surface，可以控制这个 Surface 的格式和尺寸，以及绘制位置，通常与 MediaPlayer 或 MediaRecorder 类结合使用，用于提供一个播放视频的预览窗口。

SurfaceView 变得可见时，Surface 被创建；SurfaceView 隐藏前，Surface 被销毁。这



样能节省资源。可以通过 SurfaceHolder 接口访问这个 Surface，调用 getHolder()方法可以得到这个接口。Surface 是纵深排序(Z-ordered)的，它总在自己所在窗口的后面。SurfaceView 提供了一个可见区域，只有在这个可见区域内的 Surface 部分内容才可见，可见区域外的部分不可见。Surface 的排版显示受到视图层级关系的影响，它的兄弟视图节点会在顶端显示。这意味着 Surface 的内容会被它的兄弟视图遮挡，这一特性可以用来放置遮盖物(overlays)，例如，文本和按钮等控件。但是要注意，如果 Surface 上面有透明控件，那么它的每次变化都会引起框架重新计算它和顶层控件的透明效果，这会影响性能。

SurfaceView 默认使用双缓冲技术，它支持在子线程(渲染线程)中绘制图像，这样就不会阻塞主线程了，所以它非常适合于游戏的开发。一般来说，所有 SurfaceView 和 SurfaceHolder. Callback 的方法都应该在 UI 线程里调用，一般来说就是应用程序主线程。子线程所要访问的各种变量应该作同步处理。由于 surface 可能被销毁，它只在 SurfaceHolder. Callback. surfaceCreated()和 SurfaceHolder. Callback. surfaceDestroyed()之间有效，所以要确保渲染线程访问的是合法有效的 surface。

初始的 SurfaceView 将决定视频的播放大小。系统会自动适配 SurfaceView 和 Video 的大小比例，使之恰当。

使用 SurfaceView 的一般过程：首先继承 SurfaceView，并实现 SurfaceHolder. Callback 接口，实现它的 3 个方法：surfaceCreated()、surfaceChanged()和 surfaceDestroyed()。还需要获得 SurfaceHolder，并添加回调函数，这样这 3 个方法才会执行。

surfaceCreated(SurfaceHolder holder)方法在 surface 创建时调用，一般在该方法中启动绘图的线程。surfaceChanged(SurfaceHolder holder, int format, int width, int height)方法在 surface 尺寸发生改变时调用，如横竖屏切换。surfaceDestroyed(SurfaceHolder holder)方法在 surface 被销毁时调用，一般在该方法中停止绘图线程。

9.1.4　MediaPlayer 和 MediaRecorder 类

MediaPlayer 和 MediaRecorder 类分别用于播放/录制音频、视频文件。

1. MediaPlayer

MediaPlayer 类用于播放音频、视频文件，提供了对音频、视频文件操作的一些重要方法，如播放、停止、暂停和重复播放等。而播放的音频、视频文件则可以来自于 raw 源文件、本地文件系统和通过网络传送的文件流。

MediaPlayer 的运行是基于状态的。当一个 MediaPlayer 对象被刚刚用 new 操作符创建或是调用了 reset()方法后，它就处于 Idle(空闲)状态。当调用了 release()方法后，它就处于 End(结束)状态。这两种状态之间是 MediaPlayer 对象的生命周期。MediaPlayer 的状态及其转换如图 9-3 所示。

1) Idle(空闲)状态

使用 new 方法创建一个新的 MediaPlayer 对象，或调用 reset()方法，它处于 Idle

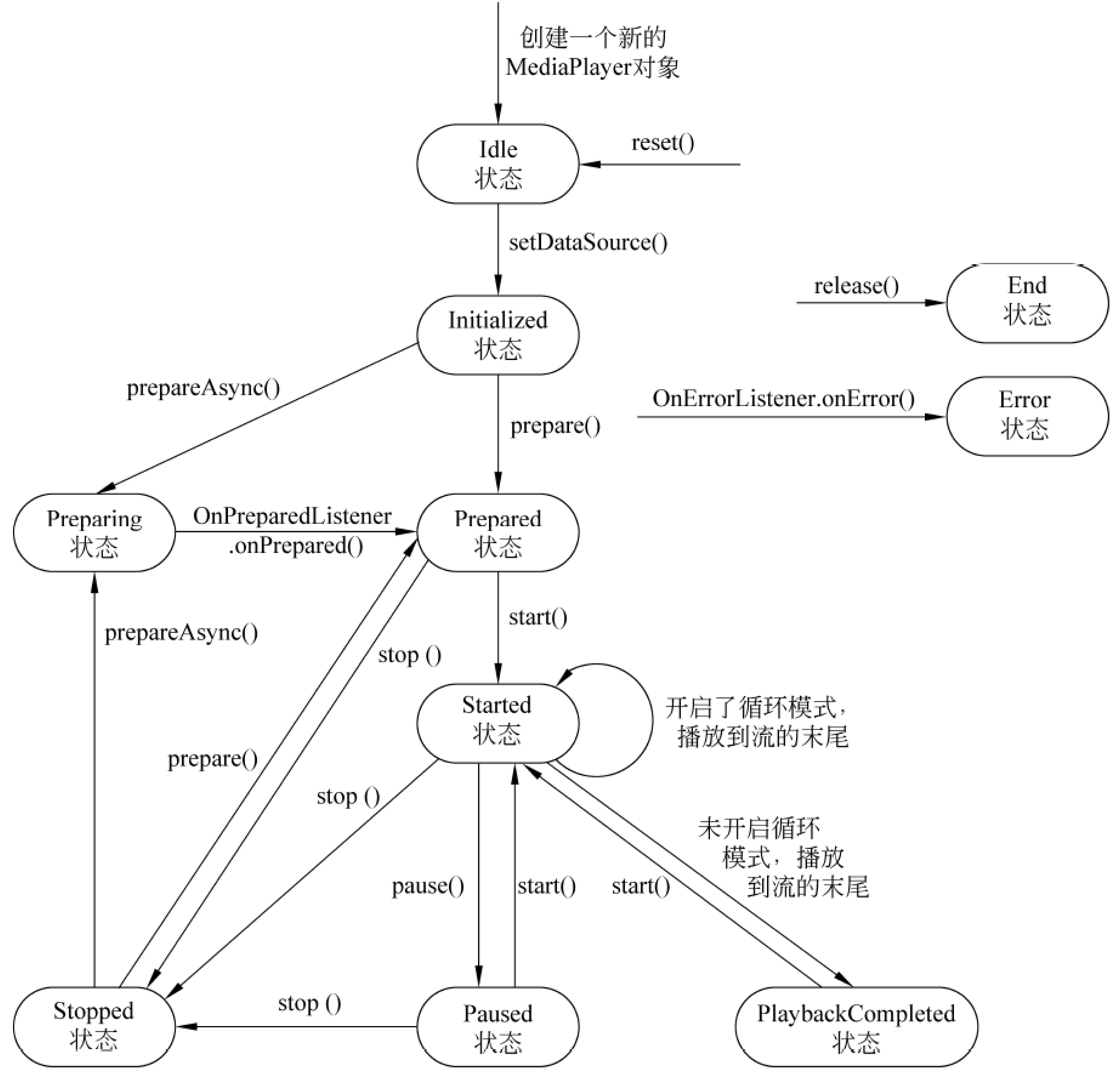


图 9-3 MediaPlayer 的生命周期

状态。

2）Initialized(已初始化)状态

调用 setDataSource(FileDescriptor)、setDataSource(String)、setDataSource(Context,Uri),或 setDataSource(FileDescriptor,long,long)方法会使处于 Idle 状态的对象迁移到 Initialized 状态。

注意：若当 MediaPlayer 对象处于其他状态(非 Idle 状态)下,调用 setDataSource()方法,会抛出 IllegalStateException 异常。

3）Prepared(就绪)状态

在开始播放之前,MediaPlayer 对象必须要进入 Prepared 状态。

有两种方法(同步和异步)可以使 MediaPlayer 对象进入 Prepared 状态：调用 prepare()方法(同步),此方法返回就表示该 MediaPlayer 对象已经进入了 Prepared 状态;或调用 prepareAsync()方法(异步),此方法会使此 MediaPlayer 对象进入 Preparing 状态并返回,而内部的播放引擎会继续未完成的准备工作。当同步版本返回时或异步版本的准备工作完全完成时就会调用客户端提供的 OnPreparedListener.onPrepared()监

听方法。可以调用 MediaPlayer. setOnPreparedListener(android. media. MediaPlayer. OnPreparedListener)方法来注册 OnPreparedListener。

在不合适的状态下调用 prepare()和 prepareAsync()方法会抛出 IllegalStateException 异常。当 MediaPlayer 对象处于 Prepared 状态的时候,可以调整音频、视频的属性,如音量,播放时是否一直亮屏,循环播放等。

4) Started(播放)状态

要开始播放,必须调用 start()方法。当此方法成功返回时,MediaPlayer 的对象处于 Started 状态。可以调用 isPlaying()方法来测试某个 MediaPlayer 对象是否在 Started 状态。

当处于 Started 状态时,内部播放引擎会调用客户端程序提供的 OnBuffering-UpdateListener. onBufferingUpdate()回调方法,此回调方法允许应用程序追踪流播放的缓冲的状态。

对一个已经处于 Started 状态的 MediaPlayer 对象调用 start()方法将不会执行任何操作。

5) Paused(暂停)状态

播放可以被暂停、停止,以及调整当前播放位置。当调用 pause()方法并返回时,会使 MediaPlayer 对象进入 Paused 状态。Started 与 Paused 状态的相互转换在内部的播放引擎中是异步的,所以可能需要一点时间在 isPlaying()方法中更新状态,若正在播放流内容,这段时间可能会有几秒钟。

调用 start()方法会让一个处于 Paused 状态的 MediaPlayer 对象从之前暂停的地方恢复播放。当调用 start()方法返回的时候,MediaPlayer 对象的状态会又变成 Started 状态。

对一个已经处于 Paused 状态的 MediaPlayer 对象调用 pause()方法将不会执行任何操作。

6) Stopped(停止)状态

调用 stop()方法会停止播放,并且还会让一个处于 Started、Paused、Prepared 或 PlaybackCompleted 状态的 MediaPlayer 进入 Stopped 状态。

对一个已经处于 Stopped 状态的 MediaPlayer 对象调用 stop()方法将不会执行任何操作。

调用 seekTo()方法可以调整播放的位置。seekTo(int)方法是异步执行的,所以它可以马上返回,但是实际的定位播放操作可能需要一段时间才能完成,尤其是播放流形式的音频、视频。seekTo(int)方法可以在其他状态下调用,比如 Prepared、Paused 和 PlaybackCompleted 状态。此外,当前的播放位置,可以通过调用 getCurrentPosition()方法得到,它可以用于帮助播放应用程序不断更新播放进度

7) PlaybackCompleted(播放完成)状态

当播放到流的末尾,播放就完成了。如果调用 setLooping(boolean)方法开启了循环模式,那么这个 MediaPlayer 对象会重新进入 Started 状态。如果没有开启循环模式,那么内部的播放引擎会调用客户端程序提供的 OnCompletion. onCompletion()回调方法。

可以通过调用 MediaPlayer. setOnCompletionListener(OnCompletionListener)方法来设置。内部的播放引擎一旦调用了 OnCompletion. onCompletion()回调方法,说明这个 MediaPlayer 对象进入 PlaybackCompleted 状态。

当处于 PlaybackCompleted 状态的时候,可以再调用 start()方法来让这个 MediaPlayer 对象再次进入 Started 状态。

8) Error(错误)状态

一旦发生错误,MediaPlayer 对象会进入到 Error 状态。可以调用 reset()方法来把这个对象恢复成 Idle 状态。在不合法的状态下调用一些方法,如 prepare()、prepareAsync()和 setDataSource()方法会抛出 IllegalStateException 异常,使其进入 Error 状态。

9) End(结束)状态

当调用 release()方法后,它就处于 End 状态。一旦一个 MediaPlayer 对象不再被使用,应立即调用 release()方法来释放在内部的播放引擎中与这个 MediaPlayer 对象关联的资源。资源可能包括如硬件加速组件的单态组件,若没有调用 release()方法可能会导致之后的 MediaPlayer 对象实例无法使用这种单态硬件资源,导致运行失败。一旦 MediaPlayer 对象进入 End 状态,它不能再被使用,也没有办法再迁移到其他状态。

特定的操作只能在特定的状态时才有效,所以编写程序时必须时刻注意它的变化。如果在错误的状态下执行一个操作,系统可能抛出一个异常或导致一个意外的行为。例如,当创建一个新的 MediaPlayer 对象,它处于 Idle 状态,此时,应调用 setDataSource()方法初始化,使它进入 Initialized 状态。之后,应使用 prepare()方法或 prepareAsync()方法准备它。当 MediaPlayer 准备完成,它将进入 Prepared 状态,这表示可以调用start()方法来播放。当调用 stop()方法后,注意不能再调用 start()方法,除非使其重新进入 Prepared 状态。

2. MediaRecorder

MediaRecorder 用于录制音频、视频。与 MediaPlayer 类似,它的运行是也基于状态的。MediaRecorder 主要有以下 7 个状态。

1) Initial(初始)状态

使用 new 方法创建一个新的 MediaRecorder 对象,它处于 Initial 状态。

2) Initialized(已初始化)状态

初始状态时,在设定视频源或者音频源之后将转换为 Initialized 状态。调用 reset()方法可以转换成 Initial 状态。

3) DataSourceConfigured(数据源配置)状态

已初始化状态时,可以通过设置输出格式转换为 DataSourceConfigured 状态。这期间可以设定编码方式、输出文件、屏幕旋转、预览显示等。它仍然可以通过调用 reset()方法回到 Initial 状态。

4) Prepared(就绪)状态

DataSourceConfigured 状态下,通过调用 prepare()方法进入到 Prepared 状态。在就

绪状态仍然可以通过调用 reset()方法回到 Initialized 状态。

5）Recording（录制）状态

在 Prepared 状态下，通过调用 start()方法可以进入 Recording 状态。它可以通过停止或者重新启动回到 Initial 状态。

6）Released（释放）状态

可以通过调用 release()方法来进入这个状态，这时将会释放所有和 MediaRecorder 对象绑定的资源。

7）Error（错误）状态

当错误发生的时候进入 Error 状态，可以调用 reset()方法来把这个对象恢复成 Initial 状态。

9.1.5　VideoView

android.widget.VideoView 类继承自 android.view.SurfaceView 类，实现了 MediaController.MediaPlayerControl 接口，用于播放视频和播放过程的控制。它的使用过程比利用 SurfaceView 结合 MediaPlayer 播放视频更直接、更简单。

VideoView 类可以从资源文件或内容提供器等不同的来源读取视频图像，并能计算和维护视频的画面尺寸以使其适用于任何布局管理器，并提供了一些诸如缩放、着色之类的显示选项。

VideoView 的构造方法主要有 3 个。

1. public VideoView（Context context）

创建一个默认属性的 VideoView 实例。参数 context 是视图运行的应用程序上下文，通过它可以访问当前主题、资源等。

2. public VideoView（Context context，AttributeSet attrs）

创建一个带有 attrs 属性的 VideoView 实例。参数 context 是视图运行的应用程序上下文，attrs 是用于视图的 XML 标签属性集合。

3. public VideoView（Context context，AttributeSet attrs，int defStyle）

创建一个带有 attrs 属性，并且指定其默认样式的 VideoView 实例。参数 defStyle 是应用到视图的默认风格，如果其值为 0 则不应用(包括当前主题中的)风格。

VideoView 提供了一些公共方法用于播放过程的控制，例如调用start ()方法开始播放视频文件、pause()方法使播放暂停、seekTo()方法设置播放位置、setVideoPath()和 setVideoURI()方法设置视频文件的路径等。VideoView 还提供了一些方法用于获得播放过程的各类参数，例如，调用 getCurrentPosition()方法可以获得当前的播放位置、调用 getDuration()方法可以获得所播放视频的总时间、调用 isPlaying ()方法可以判断是否正在播放视频等。

9.2 摄取和使用图片

Android 系统中含有一个媒体库，其中包括存储器中所有的 Image、Video、Audio 数据。摄取和录制的图片、声音、视频等都可以存储在媒体库中。

9.2.1 利用 Camera 类实现图片的摄取

Camera 是 Android 系统定义的摄像头类，它可以用于图像预览、捕获图片和录制视频等。在照相时，这个类也可以用来设置摄像头参数。

Camera 类的常用方法有 7 个。

(1) open()：获取 Camera 实例。

(2) getParameters()：获取 Camera 的参数。

(3) setParameters(param)：设置 Camera 的参数。

(4) setPreviewDisplay(holder)：Camera 与 SurfaceHolder 联系起来，设置预览窗口。

(5) release()：释放 Camera。

(6) startPreview()：启动预览功能。

(7) stopPreview()：停止预览。

Camera 对象中含有一个内部类 Camera.Parameters，利用该类可以对 Camera 的参数进行设置。可以调用 getParameters()方法获得 Camera 的默认设置参数 Camera.Parameters，更改后用 setParameters(Camera.Parameters)方法对 Camera 重新进行设置。由于不同的设备 Camera 的参数是不同的，所以在设置时，需要首先判断设备对应的参数，再加以设置。例如，在调用 setEffects()方法之前最好先调用 getSupportedColorEffects()方法判断设备支持的参数，如果设备不支持颜色特性，那么该方法将返回一个 null。

拍照过程中的预览功能需要一个用来存放取景器的容器，这个容器就是 SurfaceView。使用 SurfaceView 的同时，还需要使用到 SurfaceHolder。SurfaceHolder 相当于一个监听器，可以监听 Surface 上的变化，通过其内部类 CallBack 来实现。

利用 Camera 摄取图片的一般步骤如下。

步骤 1：在 AndroidManifest.xml 配置文件中注册相关权限。

如果要在应用程序中使用 Camera，必须在 AndroidManifest.xml 配置文件中注册相应的 Camera 权限。如果用到了 Camera 和 auto-focus 特征，还应该设置 android.hardware.camera 和 android.hardware.camera.autofocus 权限。格式如下：

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

步骤 2：建立相机的预览功能。

使用 SurfaceView 作为取景的容器和预览窗口，并通过调用 SurfaceView 的 getHolder()方法获得其控制器 SurfaceHolder。SurfaceHolder 是系统提供的控制 SurfaceView 的控制器，通过其内部类 SurfaceHolder. Callback 来实现监听变化。Camera 可以通过调用 setPreviewDisplay(SurfaceView)方法来设置 camera 的预览窗口。例如：

```
surfaceView= (SurfaceView) findViewById(R.id.myCameraView);
SurfaceHolder myholder= surfaceView.getHolder();
```

步骤 3：在得到了 SurfaceHolder 实例对象后，通过 SurfaceHolder 对象的 addCallBack()方法，实现将 SurfaceView 的回调接口 SurfaceHolder. Callback 绑定在 SurfaceHolder 上的功能。这个接口必须要重写 3 个方法，它们是 SurfaceCreated()、SurfaceChanged()和 SurfaceDestroyed()，这 3 个方法分别在当 surfaceView 被创建后、当 surfaceView 发生变化时、当 surfaceView 销毁时调用。

步骤 4：设定 SurfaceHolder 的类型，这可以通过调用 SurfaceHolder 的 setType()方法来实现。为了实现照片预览功能，需要将 SurfaceHolder 的类型设置为 PUSH，示例如下：

```
myholder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

步骤 5：设置相机的预览窗口完成后，就可以获得 Camera 实例进行预览。

具体方法是重写 SurfaceHolder. Callback 的 SurfaceCreated()方法。当 SurfaceView 创建时，该方法被调用。通过 Camera 的静态方法 open()可以获得 Camera 实例，然后设置 Camera 的预览窗口，最后通过调用 Camera 的 startPreview()方法开始预览。例如：

```
public void surfaceCreated(SurfaceHolder holder) {
    myCamera= Camera.open();                    //获取 Camera 实例
    try {
        myCamera.setPreviewDisplay(holder);
    } catch(Exception e) {
        e.printStackTrace();
        myCamera.release();                     //如果出现异常,则释放 Camera 对象
    }
    myCamera.startPreview();                    //启动预览功能
}
```

步骤 6：在拍照结束且不需要预览时，可以调用 stopPreview()方法停止预览，同时也要调用 release()方法将 Camera 实例销毁，这些一般在前面提到的 SurfaceDestroyed()方法中实现。

步骤 7：摄取照片。调用 Camera 的 takePicture()方法可以完成拍照，获取图片。同时还需要实现 Camera. PictureCallBack 接口，重写 onPictureTaken()方法处理获取的图片。

takePicture()方法中的参数 shutter 是 Camera. ShutterCallback 类型数据，是相机快

门的回调方法，主要目的是提醒用户照片已经拍照完毕；参数 jpeg 是 Camera.PictureCallback 类型数据，是相机拍照数据的处理回调方法参数之一，用来将数据流转化为指定的图片格式并进一步处理。根据需要可将照片保存到媒体库、放到 Activity 中回显或做其他处理。

9.2.2 利用系统自带的 Camera 应用实现图片的摄取

通过调用 Android 系统自带的 Camera 应用也可以摄取图像。这时只需要指定一个 MediaStore.ACTION_IMAGE_CAPTURE 的 Action 来启动 Camera 应用即可。

【例 9-3】 工程 09_CameraExample 演示了使用前述两种方法实现照片摄取的功能。



图 9-4 示例工程的主界面

示例工程的主界面如图 9-4 所示，单击第一个按钮，切换到 MyCameraActivity；单击第二个按钮，切换到 UseCameraActivity。MyCameraActivity 类采用基于 Camera 类的方法实现拍照，主要实现了预览、单击预览图片时摄取照片、将照片存储在媒体库中。UseCameraActivity 类通过调用 Android 系统自带的 Camera 应用实现摄取照片，同样将照片存储在媒体库中。

在 MyCameraActivity 类中，将 Camera 的生命周期和 SurfaceView 的生命周期保持一致，SurfaceView 创建时，创建 Camera 实例，SurfaceView 销毁时销毁 Camera 实例。该 Activity 运行后，首先显示预览画面，单击预览画面就会获取照片并保存。获取照片使用 Camera 的 takePicture()方法实现。

使用 Camera 的 takePicture()方法获取照片，必须要实现 Camera.PictureCallBack 接口并重写其 onPictureTaken()方法。在该方法中实现获取照片的处理。本例中，在 onPictureTaken()方法中实现了照片摄取后的保存功能，照片存储为 jpg 格式，存储在媒体库中。

MyCameraActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MyCameraActivity extends Activity {
    private SurfaceView surfaceView;
    private Camera myCamera= null;            //android.hardware.Camera 对象
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.camera);
        this.setTitle("基于 Camera 的照相功能示例");
        surfaceView= (SurfaceView)this.findViewById(R.id.myCameraView);
        surfaceView.setFocusable(true);
        surfaceView.setFocusableInTouchMode(true);
        surfaceView.setClickable(true);
```

```
surfaceView.setOnClickListener(new View.OnClickListener() {
            //处理预览画面的单击事件
    public void onClick(View v) {
        Camera.ShutterCallback shutter=new ShutterCallback() {
            //相机的快门回调接口
            public void onShutter() {
            }
        };
        PictureCallback jpeg=new PictureCallback() {
            public void onPictureTaken(byte[] data, Camera camera) {
                        //参数 data 是获取的图像数据
                Uri imageUri=MyCameraActivity.this.getContentResolver().insert(MediaStore.
                Images.Media.EXTERNAL_CONTENT_URI, new ContentValues());
                try {
                    OutputStream os = MyCameraActivity. this. getContentRe-solver ( ).
                    openOutputStream(imageUri);
                            //获取输出流
                    os.write(data);         //摄取的图像数据写入输出流
                    os.flush();
                    os.close();
                } catch(Exception e) {
                    e.printStackTrace();
                }
                myCamera.startPreview();      //保存图片后,再次回到预览状态
            }
        };
        myCamera.takePicture(shutter, null, jpeg);
    }
});
SurfaceHolder myholder=surfaceView.getHolder();
myholder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    //为了实现照片预览功能,需要将 SurfaceHolder 的类型设置为 PUSH
myholder.addCallback(new Callback() {
    //设置回调函数,SurfaceHolder.Callback
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    }
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        //当 Surface 被创建的时候,该方法被调用,可以在这里实例化 Camera 对象
        myCamera=Camera.open();                //获取 Camera 实例
        try {
            Parameters pa=myCamera.getParameters();
            pa.setPictureFormat(ImageFormat.JPEG);
```

```
                pa.setPreviewSize(480,320);
                myCamera.setParameters(pa);
                myCamera.setPreviewDisplay(holder);                    //设置预览窗口
                myCamera.startPreview();                               //开始预览
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            myCamera.stopPreview();
            myCamera.release();
            myCamera=null;
        }
    });
  }
}
```

UseCameraActivity 类调用 Android 系统自带的 Camera 应用，实现了照片的摄取、保存并回放显示。主要代码如下：

```
//package 和 import 语句略
public class UseCameraActivity extends Activity implements OnClickListener{
    private ImageView imageView;
    private Uri imageUri;
    Button Btn1;
    private TextView text;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        this.setTitle("调用系统自带的 Camera 应用摄取图像示例");
        text= (TextView)this.findViewById(R.id.text);
        imageView= (ImageView)this.findViewById(R.id.myimage);
        Btn1= (Button)this.findViewById(R.id.btn_capture);
        Btn1.setOnClickListener(this);
    }
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if(resultCode= =RESULT_OK) {
            imageUri=data.getData();
            text.setText("拍摄照片:"+imageUri.toString());
            Btn1.setText("继续拍摄");
            imageView.setImageURI(imageUri);
        }
    }
```




```
    public void onClick(View v) {
        int id=v.getId();
        if(id==R.id.btn_capture) {
            Intent intent=new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            startActivityForResult(intent, 1);
                //调用系统自带的 Camera 应用,拍摄的照片自动存储到媒体库中
        }
}
```

单击按钮会通过调用 startActivityForResult()方法启动 Android 系统自带的 Camera 应用,屏幕显示摄像头预览的画面。拍摄完成后返回到 UseCameraActivity 界面,获取到拍摄照片存储的 Uri 路径后将其显示在一个 ImageView 控件中。拍摄完一张照片后的程序界面如图 9-5 所示。



图 9-5　调用系统自带的 Camera 应用摄取图像

注意：由于涉及 Camera 硬件的支持,示例程序在真实设备上才能正常运行。

9.2.3　检索并显示媒体库中的图片

Android 系统以 ContentProvider 的方式共享媒体库的数据,所以要通过调用 ContentResolver 的 query()方法获取媒体库中的图片数据。

如果是使用模拟器运行程序,在使用媒体库之前,要先向在模拟器上的 SDCard 中加入图片文件。方法是在模拟器已经启动的情况下,切换到 DDMS 视图的 File Explorer 面板,如图 9-6 所示。在面板中选择文件夹 storage/sdcard/Picture,单击右上角的 Push a file onto the device 按钮,即可以将图片文件复制到模拟器的 SDCard 中。

Android 是在系统启动的时候来扫描模拟器上 SDCard 中多媒体文件的,对于刚刚添加的图片,媒体库是没有办法获取到的。所以添加了图片之后,需要将模拟器关闭,再重

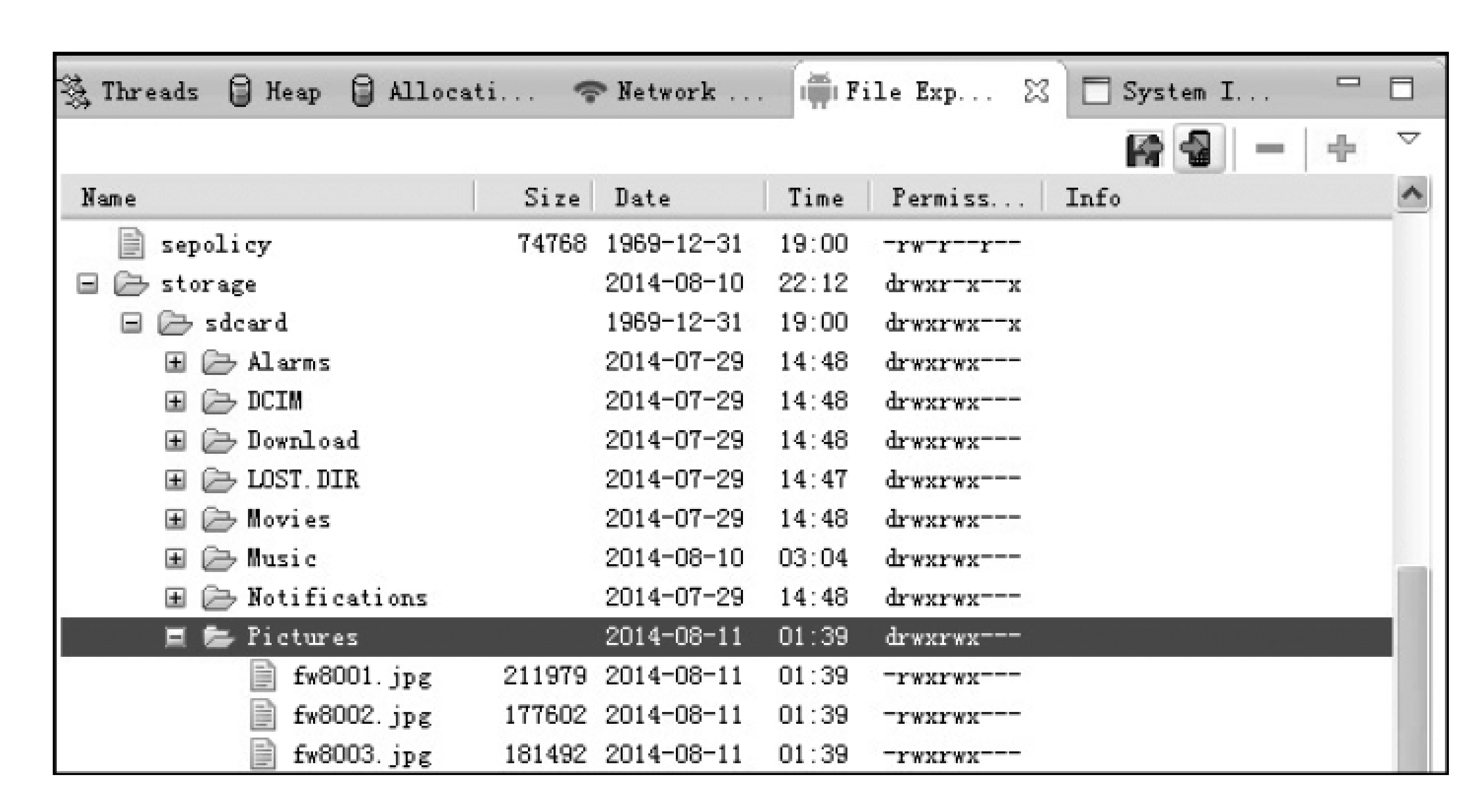


图 9-6　File Explorer 面板

新启动一下。

【例 9-4】 工程 09_ImageManageExample 演示了检索并显示媒体库中的图片。

示例工程的实现过程如下。

步骤 1：新建工程项目 09_ImageManageExample，定义 Activity_main.xml 文件。本例中使用 ImageButton 控件来显示媒体库中的图片，单击图片可以显示下一张图片。另外使用两个按钮控制图片的向前和向后切换。

步骤 2：定义 MainActivity 类，从媒体库中取出图片显示到 ImageButton 控件中。MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    public static final float DISPLAY_WIDTH= 300;
    public static final float DISPLAY_HEIGHT= 200;
    private Button BtnPre,BtnNext;
    private ImageButton imageView;
    private TextView nameView;
    private Cursor cursor;
    private String imagePath;                    //存放某张图片的路径
    private Bitmap image;
    private int imageIndex,nameIndex,titleIndex;
    //这 3个变量用来保存 Media.DATA、Media.TITLE 和 Media.DISPLAY_NAME 的索引号
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imageView= (ImageButton)this.findViewById(R.id.Ibtn_image);
        imageView.setOnClickListener(clickListener);
        nameView= (TextView)this.findViewById(R.id.Tv_name);
        BtnPre= (Button)findViewById(R.id.btn_pre);
        BtnNext= (Button)findViewById(R.id.btn_next);
        String columns[]= new String[]{Media.DATA,Media._ID,Media.TITLE,Media.DISPLAY_NAME };
```

```
        cursor=this.getContentResolver().query(Media.EXTERNAL_CONTENT_URI, columns, null, null,
        null);
            //检索媒体库中的图像数据,包括图片的路径、名称和标题等
        imageIndex=cursor.getColumnIndexOrThrow(Media.DATA);
        titleIndex=cursor.getColumnIndexOrThrow(Media.TITLE);
        nameIndex=cursor.getColumnIndexOrThrow(Media.DISPLAY_NAME);
        if(cursor.moveToFirst()) {                    //判断 Cursor 是否有值
            showImage();                              //调用自定义方法,显示第一张图片
        }
        else{
            Toast.makeText(this,"媒体库中没有图片!", Toast.LENGTH_SHORT).show();
        }
        BtnPre.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                readPre();                            //调用自定义方法,读取前一张图片
            }
        });
        BtnNext.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                readNext();                           //调用自定义方法,读取后一张图片
            }
        });
    }
    private void readPre() {                          //自定义方法,读取前一张图片
        cursor.moveToPrevious();
        if(!cursor.isBeforeFirst()) {
            showImage();
        }
        else {
            Toast.makeText(this,"已经是第一个图片!", Toast.LENGTH_SHORT).show();
            cursor.moveToNext();
        }
    }
    private void readNext() {                         //自定义方法,读取后一张图片
        …                                             //实现代码与 readPre()方法类似,略
    }
        //图片显示在 ImageButton 控件上,单击图片,显示下一张图片
    private View.OnClickListener clickListener=new View.OnClickListener() {
        public void onClick(View v) {
            if(cursor.moveToNext()) {
                showImage();
            }
        }
    };
```

```
    private void showImage() {                          //自定义方法,显示一张图片
        imagePath=cursor.getString(imageIndex);         //获取图片存储的位置信息
        image=decodeBitmap(imagePath);                  //获取图片数据
        imageView.setImageBitmap(image);                //图片显示到 ImageButton 控件中
        nameView.setText("图片信息:"+cursor.getString(nameIndex)+":"+cursor.getString
        (titleIndex));
                          //获取图片信息,并显示到 TextView 控件中
    }
    private Bitmap decodeBitmap(String path) {          //从 path 中获取图片信息
        BitmapFactory.Options op=new BitmapFactory.Options();
        op.inJustDecodeBounds=true;
        Bitmap bmp=BitmapFactory.decodeFile(path, op);                  //获取图片数据
        int wRatio=(int)Math.ceil(op.outWidth/DISPLAY_WIDTH);
        int hRatio=(int)Math.ceil(op.outHeight/DISPLAY_HEIGHT);
            //如果超出指定大小,则缩小图片尺寸与显示区域尺寸的比例
        if(wRatio>1 && hRatio>1) {
            if(wRatio>hRatio) {
                op.inSampleSize=wRatio;
            }else{
                op.inSampleSize=hRatio;
            }
        }
        op.inJustDecodeBounds=false;
        bmp=BitmapFactory.decodeFile(path, op);
        return bmp;
    }
}
```

步骤 3：在 AndroidManifest.xml 配置文件中注册允许读外部存储的权限：

```
<uses-permission android:name="android.
permission.READ_EXTERNAL_STORAGE" />
```

程序的运行结果如图 9-7 所示。

图 9-7 示例程序的运行结果

9.3 音频文件的播放

9.3.1 使用 Android 系统自带的播放器

Android 系统有其自带的播放器,可以使用隐式 Intent 来调用它。通过指定一个 ACTION_VIEW 的 Action,同时用一个 Uri 来指定要播放文件的路径,并指定所要播放文件的格式信息(MIME),就可以调用播放器来播放该音频文件了。

【例 9-5】 示例工程 09_MusicPlayerExample 演示了如何利用 Android 自带的播放

器来播放音频文件。

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent intent=new Intent(Intent.ACTION_VIEW);
        File sdcard=Environment.getExternalStorageDirectory();
            //获取 sdcard 路径
        File audioFile=new File(sdcard.getPath()+"/music/music01.mp3");
            //获取音频文件的 Uri
        Uri audioUri=Uri.fromFile(audioFile);
            //指定 Uri 和 MIME
        intent.setDataAndType(audioUri, "audio/mp3");
        startActivity(intent);                  //播放
    }
}
```

9.3.2　使用 MediaPlayer 类播放音频文件

可以利用 android.media.MediaPlayer 类来播放音频文件。在使用 MediaPlayer 之前，必须在 AndroidManifest.xml 配置文件中注册相关的权限。如果应用程序在播放过程中需要阻止屏幕变暗或阻止处理器睡眠，或使用 MediaPlayer.setScreenOnWhilePlaying()和 MediaPlayer.setWakeMode()方法，必须注册对睡眠加锁的权限，权限名称为 android.permission.WAKE_LOCK；如果使用 MediaPlayer 来播放网络流中的内容，还必须注册网络存取权限，权限名称为 android.permission.INTERNET。

1. 播放 raw 资源文件

raw 文件夹中一般存放原始资源文件，音频文件即是其中的一种。Android 系统不会解析 raw 资源文件，它必须是一种适当编码和格式化的媒体文件。

从 raw 源文件中播放多媒体文件按照以下步骤实现。

步骤 1：将多媒体音频文件放入 raw 文件夹中。如果在工程中没有这个文件夹，可以在工程中新建一个 res/raw 文件夹，并将相应的音频文件复制到文件夹中。

步骤 2：声明 MediaPlayer 对象，调用 MediaPlayer 的静态方法 create()创建 MediaPlayer 对象并与指定的多媒体文件关联。

步骤 3：调用 MediaPlayer 对象的 start()方法播放指定的多媒体音频文件。

针对上述步骤的示例代码如下：

```
MediaPlayer mp=MediaPlayer.create(this, R.raw.musicname);
                            //创建 MediaPlayer 对象 mp 并关联 musicname 音频文件
```

```
mp.start();                                    //播放音频文件
```

2. 播放本地文件

如果想从本地文件系统中播放，需要让播放器知道文件所在位置。Mediaplayer 对象的 setDataSource()方法可以实现这一功能。从本地文件系统中播放音频文件的一般步骤如下。

步骤 1：用 MediaPlayer 的默认构造方法构造一个 MediaPlayer 对象。

步骤 2：调用 MediaPlayer 对象的 setDataSource()方法将文件的路径传入，指定本地音频文件所在的位置。

步骤 3：调用 prepare()方法进行播放前的准备。

步骤 4：调用 start()方法播放。

示例代码片段如下：

```
MediaPlayer mp=new MediaPlayer();              //构造一个 MediaPlayer 对象 mp
mp.reset();                                    //媒体播放器重置为空状态
try {
    mp.setDataSource("/storage/sdcard/music/music01.mp3");
                                               //设置音频文件的路径
    mp.prepare();                              //播放前的准备
} catch(IllegalArgumentException e) {
        e.printStackTrace();
} catch(IllegalStateException e) {
        e.printStackTrace();
} catch(IOException e) {
        e.printStackTrace();
}
mp.start();                                    //开始播放
```

需要注意的是，当使用 setDataSource()时必须捕获和传递 IllegalArgumentException 和 IOException，因为引用的文件可能不存在。另外，如果播放的文件在 SD 卡中，还需要在 AndroidManifest.xml 配置文件中注册外存的读权限。

3. 播放网络文件

MediaPlayer 可以实现播放网络中的音频文件。具体方法是，通过创建网络 URI 实例，调用 MediaPlayer 的静态方法 create()，通过传递 URI 参数来完成音频文件的播放。也可以通过调用 MediaPlayer 的 setDataSource()方法，设置文件播放路径来完成播放。

4. 播放过程的控制

如果想停止音频文件的播放，可以调用 MediaPlayer 对象的 stop()方法停止播放；暂停播放调用 pause()方法；重复播放则需要先调用 reset()方法初始化 MediaPlayer 状态，然后调用 prepare()方法准备播放，最后调用 start()方法播放媒体文件。当暂停文件播

放后，如果要继续播放，重新调用 start()方法即可。

9.3.3　音频文件播放示例

【例 9-6】　工程 09_AudioPlayerExample 实现了一个简易的音频文件播放器。分别使用了前述的 3 种音频文件播放方式：raw 文件夹下文件的播放、本地文件播放和网络文件播放。播放过程中可以进行暂停、继续、停止播放。

MainActivity 的布局如图 9-8 所示。



图 9-8　简易播放器的界面布局

MainActivity 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity implements
OnClickListener {
    MediaPlayer mp=new MediaPlayer();
    Button BtnRaw,BtnLocal,BtnUri,BtnPau,BtnCon,BtnStop;
    TextView text;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text= (TextView)this.findViewById(R.id.text);
        BtnRaw= (Button)this.findViewById(R.id.BtnRaw);
        BtnLocal= (Button)this.findViewById(R.id.BtnLocal);
        BtnUri= (Button)this.findViewById(R.id.BtnUri);
        BtnPau= (Button)this.findViewById(R.id.BtnPau);
        BtnCon= (Button)this.findViewById(R.id.BtnCon);
        BtnStop= (Button)this.findViewById(R.id.BtnStop);
        BtnRaw.setOnClickListener(this);
        BtnLocal.setOnClickListener(this);
        BtnUri.setOnClickListener(this);
        BtnPau.setOnClickListener(this);
        BtnCon.setOnClickListener(this);
        BtnStop.setOnClickListener(this);
    }
    public void onClick(View v) {
        switch(v.getId()) {
          case R.id.BtnRaw:
              text.setText("\n播放 raw 文件\n");
              mp.reset();                          //媒体播放器重置为空状态
              mp=MediaPlayer.create(this, R.raw.music01);
                          //创建 MediaPlayer 对象 mp 并关联音频文件
              mp.start();                          //播放音频文件
              break;
          case R.id.BtnLocal:
```

```
            text.setText("\n播放本地文件:\n");
            mp.reset();                          //媒体播放器重置为空状态
            try {
                mp.setDataSource(Environment.getExternalStorageDirectory().getPath()+"/music/
                music03.mp3");                                   //设置音频文件的路径
                mp.prepare();                    //播放前的准备
             } catch(IllegalArgumentException e) {
                    e.printStackTrace();
             } catch(IllegalStateException e) {
                    e.printStackTrace();
             } catch(IOException e) {
                    e.printStackTrace();
             }
             mp.start();                         //开始播放
           break;
        case R.id.BtnUri:
            text.setText("\n播放网络文件\n");
              mp.reset();                        //媒体播放器重置为空状态
            try {
                String path="http://www.51work6.com/android_book/audio/ma_mma.mp3";
                Uri uri=Uri.parse(path);         //将路径字符串解析为 Uri 实例
                mp=MediaPlayer.create(getApplicationContext(), uri);
                                    //设置音频文件的路径
                mp.prepare();                    //播放前的准备
             } catch(IllegalArgumentException e) {
                  e.printStackTrace();
             } catch(IllegalStateException e) {
                  e.printStackTrace();
             }
             mp.start();                         //开始播放
             break;
        case R.id.BtnPau:
            text.setText("\n暂停播放\n");
            mp.pause();                          //暂停播放
            break;
        case R.id.BtnCon:
            text.setText("\n继续播放\n");
            mp.start();                          //继续播放
            break;
        case R.id.BtnStop:
            text.setText("\n停止播放\n");
            mp.stop();                           //停止播放
            break;
        default:
```

```
                break;
            }
        }
    }
```

9.4　视频文件的播放

在 Android 中,有 3 种方式来实现视频的播放:使用系统自带的播放器、使用 VideoView 播放、使用 MediaPlayer 类和 SurfaceView 控件来实现播放。下面介绍这 3 种方式的具体使用方法。

9.4.1　使用 Android 自带的播放器播放视频

使用 Android 自带的播放器,只需指定 Action 为 ACTION_VIEW,Data 为视频文件设为 Uri,Type 为其 MIME 类型,然后调用播放器即可。

【例 9-7】　工程 09_VideoPlayerExample 演示了使用系统自带播放器播放视频的方法。

MainActivity 类的主要代码如下:

```
//package 和 import 语句略
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent intent=new Intent(Intent.ACTION_VIEW);
                                                  //设置 Action
        File sdcard=Environment.getExternalStorageDirectory();
                                                  //获取 sdcard 路径
        Uri uri=Uri.parse(sdcard.getPath()+"/movies/video01.mp4");
                //获取该文件的 Uri
        intent.setDataAndType(uri, "video/mp4");
                //设置视频文件的类型
        startActivity(intent);                    //调用系统自带的播放器,开始播放
    }
}
```

示例程序运行后,调用 Android 自带的播放器全屏播放指定的视频文件,播放完成后自动回到 MainActivity 的界面。

9.4.2　使用 VideoView 播放视频

为了简化播放视频文件的处理过程,Android 框架提供了 VideoView 类来封装 MediaPalyer。VideoView 继承自 SurfaceView,使用 MediaPlayer 来播放视频。

VideoView 通过与 MediaController 类结合使用，编程者可以不用自己控制播放与暂停。

当 VideoView 创建的时候，MediaPalyer 对象将会创建；当 VideoView 对象销毁的时候，MediaPlayer 对象将会释放。不需要管理 MediaPalyer 的各种状态，这些状态都已经被 VideoView 封装了。

【例 9-8】 工程 09_VideoViewExample 使用了 VideoView 来实现播放视频。在布局文件中使用 VideoView 结合 MediaController 来实现对其控制。

主要设计步骤如下。

步骤 1：创建 09_VideoViewExample 工程，在界面布局文件中添加 VideoView 控件，或在 Java 程序中创建 VideoView 对象。本例使用前一种方法，activity_main 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"   />
    <VideoView
         android:id="@+id/video_view"
           android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_centerInParent="true"   />
</LinearLayout>
```

步骤 2：定义 MainActivity 类。在 Activity 中获取布局文件中的 VideoView 实例，调用 VideoView 类的方法来加载指定的视频文件。有两个方法可以实现这一功能，其中 setVideoPath(String path)方法用于加载 path 路径指定的视频文件，而 setVideoURI (Uri uri)方法用于加载 Uri 所对应的视频文件。可视具体情况选择其一。

步骤 3：调用 VideoView 的 start()、stop()、pause()方法来控制视频的播放。MainActivity 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity implements MediaPlayer.OnErrorListener, MediaPlayer.
OnCompletionListener {
    private VideoView myVideoView;
    private Uri mUri;
    private int mPositionWhenPaused=-1;
    private MediaController mMediaController;
    private TextView textView;
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.setTitle("使用 VideoView 播放视频文件示例");
        myVideoView= (VideoView)findViewById(R.id.video_view);
        textView= (TextView)findViewById(R.id.text_view);
        File sdcard= Environment.getExternalStorageDirectory();
                                            //获取 sdcard 路径
        mUri= Uri.parse(sdcard.getPath()+ "/movies/video0010.3gp");
                                            //将路径字符串解析为 Uri 实例
        textView.setText("播放视频:"+ mUri.toString()+ "\n");
        mMediaController= new MediaController(this);
                                            //创建媒体控制器
        myVideoView.setMediaController(mMediaController);
                                            //设置一个控制条
}

    public void onStart() {
        myVideoView.setVideoURI(mUri);      //设置视频文件的路径
        myVideoView.start();                //播放视频
        super.onStart();
    }
    //监听 MediaPlayer 上报的错误信息
     public boolean onError(MediaPlayer mp, int what, int extra) {
        return false;
     }
     //Video 播完的时候得到通知
    public void onCompletion(MediaPlayer mp) {
        this.finish();
    }
}
```

通过实现 MediaPlayer.OnErrorListener 来监听 MediaPlayer 上报的错误信息。另外，实现 MediaPlayer.OnCompletionListener 接口，将会在 Video 播完的时候得到通知，本例只是设置了在播完后结束程序。

示例程序为 VideoView 设置了一个 Android 自带的控制条，具有“暂停”、“快进”、“快退”按钮和一个进度显示条。示例程序的运行结果如图 9-9 所示。

9.4.3 使用 MediaPlayer 和 SurfaceView 播放视频

播放视频文件，更为直接的方法是使用 MediaPlayer 类实现。MediaPlayer 需要使用一个 SurfaceView 对象作为输出设备。

【例 9-9】 工程 09_SurfaceViewExample 演示了使用 MediaPlayer 和 SurfaceView 播放视频的方法。

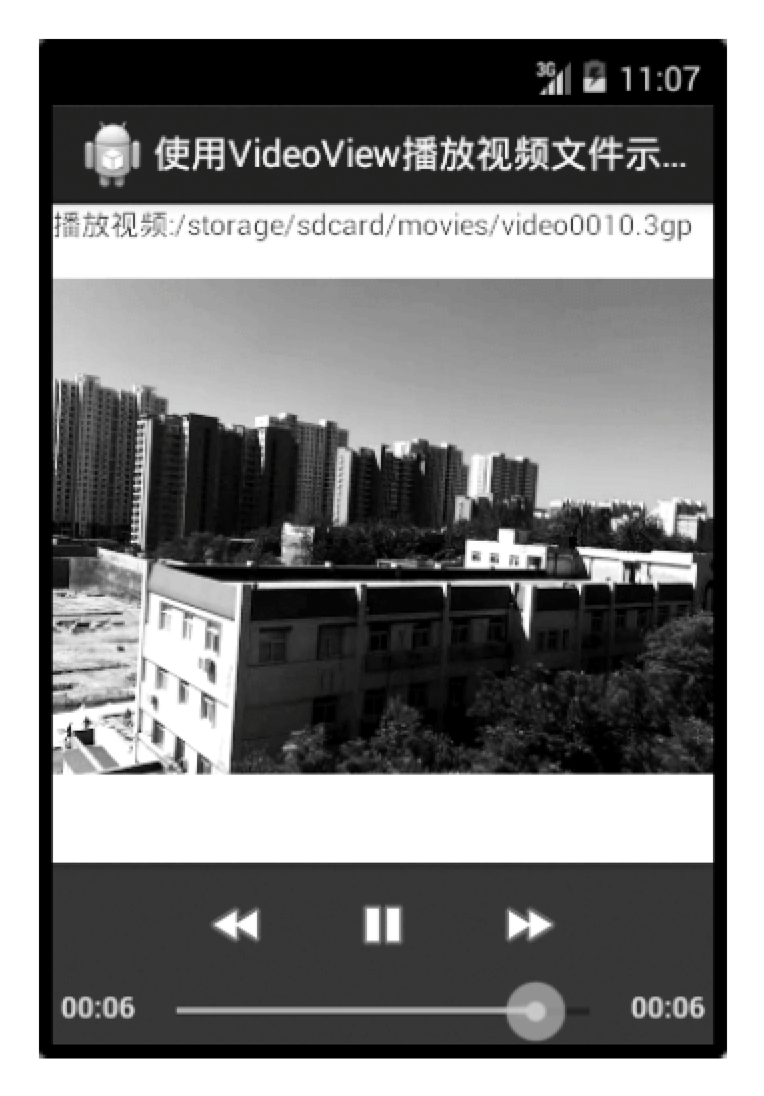


图 9-9 利用 VideoView 播放视频文件

主要实现步骤如下。

步骤 1：创建 09_VideoViewExample 工程。本例中播放 SD 卡中的视频文件，所以需要在 AndroidManifest. xml 配置文件中注册外存的读权限：

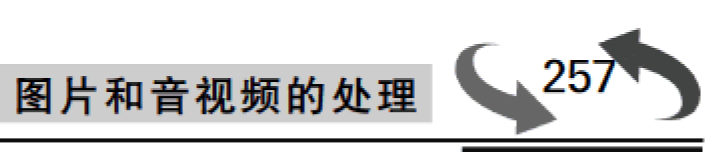
```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

步骤 2：在界面布局文件中添加 SurfaceView 控件，布局文件内容如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <SurfaceView
        android:id="@+id/myVideoView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"    />
</LinearLayout>
```

步骤 3：定义 MainActivity 类，获取布局文件中的 SurfaceView 实例，并实现 SurfaceHolder. Callback 接口，重写它的 surfaceCreated()方法，实现视频的播放。

```
//package 和 import 语句略
```




```
public class MainActivity extends Activity {
    private SurfaceView mysurfaceView;
    private SurfaceHolder myHolder=null;
    private MediaPlayer myMediaPlayer;
    private String myPath;
    private TextView text;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.surface);
        this.setTitle("使用 MediaPlayer 播放视频示例");
        text=(TextView)this.findViewById(R.id.text);
        mysurfaceView=(SurfaceView)this.findViewById(R.id.myVideoView);
        myPath=Environment.getExternalStorageDirectory().getPath()+"/movies/video0010.3gp";
                    //设置视频文件的路径
        myHolder=mysurfaceView.getHolder();
        myHolder.addCallback(new Callback() {
            @Override
            public void surfaceCreated(SurfaceHolder holder) {
                //当 Surface 被创建的时候,该方法被调用,在方法中加载视频文件并播放
                try {
                    myMediaPlayer=new MediaPlayer();
                    myMediaPlayer.setDataSource(myPath);
                    text.setText("播放视频文件:"+myPath);
                    myMediaPlayer.setDisplay(holder);
                    myMediaPlayer.prepare();
                    myMediaPlayer.start();
                } catch(Exception e) {
                }
            }
        });
    }
}
```

示例程序的运行结果如图 9-10 所示。

9.5　音频和视频的录制

9.5.1　使用 Android 系统自带的录音程序录制音频

Android 系统中有自带的音频录制程序,可以通过指定一个 Action 为 MediaStore. Audio. Media. RECORD _ SOUND _ ACTION 的 Intent 来启动它。启动时调用 startActivityForResult( ) 方法,这样录音程序返回时可以携带参数,在 onActivity-



图 9-10 利用 MediaPlayer 播放视频文件

Result()方法中，就可以获取 Intent 的 Data，其内容就是录制的音频文件对应的 URI。

【例 9-10】 工程 09_AudioRecorderExample 演示了使用 Android 系统自带的录音程序录制音频。

在 MainActivity 的界面中设置了按钮，单击按钮则调用系统自带的录音程序。MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity implements OnClickListener{
    Button Btn;
    TextView text;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Btn= (Button)this.findViewById(R.id.btn1);
    Btn.setOnClickListener(this);
    }

    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if(resultCode==RESULT_OK) {
            Uri audioPath=data.getData();          //获取到刚刚录制的音频文件的 Uri
            text.setText("\n录制的音频文件:\n "+audioPath.toString());
        }
    }

    public void onClick(View v) {                  //单击按钮事件的处理
        int id=v.getId();
```

```
            switch(id) {
            case R.id.btn1:
              Intent intent=new Intent(MediaStore.Audio.Media.RECORD_SOUND_ACTION);
              startActivityForResult(intent, 0);        //调用 Android 自带的录音程序
              break;
            }
        }
    }
```

示例程序调用 Android 自带的音频录制程序，其运行界面如图 9-11 所示。录音结束后的 Toast 提示信息如图 9-12 所示。



图 9-11　Android 自带的音频录制程序

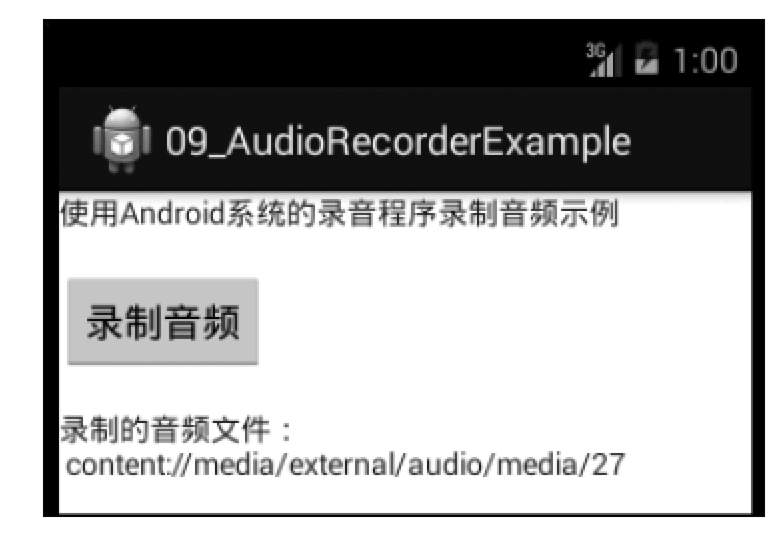


图 9-12　录音结束后的提示信息

9.5.2　使用 Android 系统自带的 Camera 应用录制视频

使用 Android 自带的 Camera 应用来录制视频过程很简单，直接指定一个 MediaStore. ACTION_VIDEO_CAPTURE 的 Action 就可以实现。

【例 9-11】　工程 09_VideoCameraExample 演示了使用系统自带的 Camera 应用录制视频、保存并回放。

MainActivity 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity implements OnClickListener{
    private VideoView videoView;
    private Uri videoUri;
    Button Btn1,Btn2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            videoView= (VideoView)this.findViewById(R.id.myVideoView);
            Btn1= (Button)this.findViewById(R.id.btn_capture);
            Btn1.setOnClickListener(this);
            Btn2= (Button)this.findViewById(R.id.btn_play);
            Btn2.setOnClickListener(this);
        }
        public void onActivityResult(int requestCode, int resultCode, Intent data) {
            if(resultCode==RESULT_OK) {
                videoUri=data.getData();              //获取到刚刚录制的视频文件的 Uri
            }
        }
        public void onClick(View v) {
            int id=v.getId();
            if(id==R.id.btn_capture) {
                Intent intent=new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
                startActivityForResult(intent, 1);    //调用 Android 自带的 Camera 应用
            }
            else if(id==R.id.btn_play) {
                videoView.setVideoURI(videoUri);
                videoView.start();                    //回放刚刚录制的视频文件
            }
        }
    }
```

由于涉及 Camera 硬件的支持，示例程序在真实设备上才能正常运行，其运行界面如图 9-13 所示。



图 9-13 示例程序的运行界面



9.5.3 使用 MediaRecorder 类录制音频和视频

使用 MediaRecorder 类可以实现音频和视频的录制功能。使用 MediaRecorder 类录制音频和视频的方法类似，所不同的是录制视频需要设置更多的参数，例如，设置用来录制视频的 Camera、设置视频图像的输出尺寸、设置视频的编码格式等。另外，视频录制需要使用 Camera，还需要在 AndroidManifest.xml 配置文件中注册相关权限。

使用 MediaRecorder 类录制音频和视频的一般步骤如下。

步骤 1：在 AndroidManifest.xml 配置文件中注册相关权限。录制音频需要获得录制 audio 的权限，视频录制需要使用 Camera 的权限，如果要将录制的文件写入 SD 卡中，则还需要外存的写入权限。

步骤 2：定义 MainActivity 类，创建 MediaRecorder 实例。用 MediaRecorder 的默认构造方法创建一个 MediaRecorder 的实例对象。

步骤 3：设置 MediaRecorder 对象的数据源。音频录制需要调用 MediaRecorder 对象的 setAudioSource()方法设置音频源。例如，下面的语句指定音频源为 MIC，即从麦克风获取音频，这是最常用的音频源。录制视频时，还可以使用音频源 MediaRecorder.AudioSource.CAMCORDER。

```
myRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

类似地，如果进行视频录制，需要调用 MediaRecorder.setVideoSource()方法设置视频源，例如，下面的语句指定从照相机采集视频：

```
myRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

步骤 4：设置音频、视频的输出格式和编码格式。下面的代码片段指定了音频编码方式为 AMR_NB，视频编码格式为 H263 格式：

```
myRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
myRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H263);
```

如果是录制视频，还需要设置视频的分辨率和视频帧率，例如：

```
myRecorder.setVideoSize(960, 544);
myRecorder.setVideoFrameRate(4);
```

MediaRecorder.OutputFormat.THREE_GPP 为 MediaRecorder 音频和视频输出格式的一种，其他的还有 AMR_NB、AMR_WB、DEFAULT、MPEG_4 和 RAW_AMR 等，详情请查看 MediaRecorder.OutputFormat 类的 API 文档。Android 支持的音频编码格式有 DEFAULT、AMR_NB、AMR_WB 和 AAC 等。详情请查看 MediaRecorder.AudioEncoder 类的 API 文档。

Android 2.2 及其以后版本采用下面方式设置输出格式和编码格式：

```
myRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));
```

步骤 5：调用 MediaRecorder. setOutputFile()设置 mediaRecorder 的输出文件名称。例如：

```
File videoFile= new File(Environment.getExternalStorageDirectory(),System.currentTimeMillis()+".
3gp");
myRecorder.setOutputFile(videoFile.getAbsolutePath());
```

在配置 MediaRecorder 的过程中，需要注意参数设置的顺序，否则应用程序可能会抛出 java. lang. IllegalStateException 异常。

步骤 6：配置完成以后，调用 MediaRecorder. prepare()准备录制。这个方法执行完后 MediaRecorder 就准备好了捕捉和编码音频及视频数据。

调用 MediaRecorder. start()方法开始录制。录制完成后，可以调用 MediaRecorder. stop()方法停止录制。当 MediaRecorder 完成音频录制，并且 MediaRecorder 不再使用时，调用 release()方法释放 MediaRecorder 的资源。

【例 9-12】 工程 09_MediaRecorder_AudioExample 演示了利用 MediaRecorder 类录制音频，并将录制的音频存储成文件。

录制音频需要获得录制 audio 的权限和向 SDCard 写数据的权限。在 AndroidManifest. xml 配置文件中添加以下代码申请。

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

定义 MainActivity 类，界面中设置 4 个按钮，单击“开始”按钮，开始录音；单击“停止”按钮，录音结束，存储录音文件；单击“播放”按钮，则播放先前录制的文件；单击“结束”按钮，则关闭 Activity，返回录制的音频的 Uri。

处理按钮单击事件的部分代码如下：

```
public void onClick(View v) {
    int id=v.getId();
    switch(id) {
    case R.id.btn_start:
        myRecorder=new MediaRecorder();            //实例化 MediaRecorder 对象
        myRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                //指定 AudioSource 为 MIC,从麦克风获取音频
        myRecorder.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);
        myRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
                //指定输出格式和编码方式
        audioFile=new File(Environment.getExternalStorageDirectory(),System.currentTimeMillis()+"
        aa.3gp");
        myRecorder.setOutputFile(audioFile.getAbsolutePath());
                //指定录音文件存储路径
        try   {
            myRecorder.prepare();                  //准备录制
        } catch(IllegalStateException e1) {
```

```
            e1.printStackTrace();
        } catch(IOException e1) {
            e1.printStackTrace();
        }
        myRecorder.start();                         //开始录制
        stateView.setText("正在录制");              //更新提示文字
        btnStart.setEnabled(false);
        btnPlay.setEnabled(false);
        btnStop.setEnabled(true);                   //更新按钮状态
        break;
    case R.id.btn_stop:
        myRecorder.stop();
        myRecorder.release();
        ContentValues values= new ContentValues();
        values.put(MediaStore.Audio.Media.TITLE, "this is my first record- audio");
        values.put(MediaStore.Audio.Media.DATE_ADDED, System.currentTime-Millis());
        values.put(MediaStore.Audio.Media.DATA, audioFile.getAbsolutePath());
        fileUri= this.getContentResolver().insert(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
        values);
                //将录制的文件存储到 MediaStore 中
        player= new MediaPlayer();                 //录制结束后,实例化 MediaPlayer 对象,准备播放
        player.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            public void onCompletion(MediaPlayer arg0) {
                stateView.setText("准备录制"); //播放结束,更新提示文字
                btnPlay.setEnabled(true);
                btnStart.setEnabled(true);
                btnStop.setEnabled(false);          //更新按钮状态
            }
        });
        try {
            player.setDataSource(audioFile.getAbsolutePath());
            player.prepare();
        } catch(IllegalArgumentException e) {
            e.printStackTrace();
        } catch(IllegalStateException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }
        stateView.setText("准备播放");              //更新提示文字
        btnPlay.setEnabled(true);
        btnStart.setEnabled(true);
        btnStop.setEnabled(false);                  //更新按钮状态
        break;
```

```
        case R.id.btn_play:
            player.start();                         //播放录音。
            stateView.setText("正在播放");
            btnStart.setEnabled(false);             //更新提示文字
            btnStop.setEnabled(false);
            btnPlay.setEnabled(false);              //更新按钮状态
            break;
        case R.id.btn_finish:
            Intent intent=new Intent();
            intent.setData(fileUri);
            this.setResult(RESULT_OK, intent);      //完成录制,返回录制的音频的 Uri
            this.finish();
            break;
        }
    }
```

示例程序的运行结果如图 9-14 所示。单击“开始”按钮,开始录音;单击“停止”按钮,录音结束,存储录音文件;单击“播放”按钮,则播放先前录制的文件。



(a)



(b)

图 9-14 录制音频文件

9.6 本 章 小 结

本章学习了 Android 系统如何处理和使用图片、音频和视频等资源。Android 系统通过媒体库管理所有 Image、Video 和 Audio 的数据,通过 MediaStore 对象可以访问媒体库中的数据。在处理和使用这些多媒体资源时可以使用 Android 系统自带的播放器和录音程序,也可以使用 MediaPlayer 类和 MediaRecorder 类。MediaPlayer 类用于播放音、视频文件,提供了对音频和视频的播放、停止、暂停、重复播放等方法;而 MediaRecorder 类用于录制音频和视频。MediaPlayer 和 MediaRecorder 的运行都是基于状态的,特定的操作只能在特定的状态时才有效,如果在错误的状态下执行一个操作,系统可能抛出一个异常或导致一个意外的行为,所以编写程序时必须时刻注意它的变化。学习本章内容时要求重点掌握图片的摄取、音频和视频的播放及录制方法,并能够编写简单的多媒体应用程序。

习　　题

1. 设计一个查看媒体库照片的程序，能显示媒体库全部照片的列表，单击列表中的某个文件即显示照片内容。

2. 设计一个照相程序，拍摄的照片即刻到另一个 Activity 中显示。

3. 设计一个音乐播放器，能播放/暂停/停止音乐，播放过程中能显示音乐文件的名称，能选择上一首/下一首音乐播放。

4. 设计一个音乐播放器，能显示媒体库全部音乐的列表，单击列表中的某个文件即开始播放，播放过程中能显示音乐文件的名称。

5. 自己设计一个录音程序，录音文件存储为/sdcard/record/myaudio.3gp。

6. 自己设计一个视频录制程序，视频文件存储为/sdcard/record/mymovie.3gp。

第10章 Web应用程序设计

Android提供了多种方式来利用 Internet资源。可以使用客户端API直接与服务器远程交互,常用的方法有利用 URLConnection、HttpURLConnection、HttpClient 或 Socket 与远程服务器交互;也可以使用 WebView 控件在 Activity 中包含一个基于 WebKit 的浏览器;或通过使用 WebService 调用远程服务器上的方法,实现特定的功能。本章主要介绍这些访问 Internet 资源的方法。

10.1 Android 网络通信概述

Android 基于 Linux 内核,它包含一组优秀的网络通信功能,提供了多个类来帮助处理网络通信。目前,Android 平台主要有 3 种网络接口可以使用,它们分别是 java.net.*(标准 Java 接口)、org.apache.*(Apache 接口)和 android.net.*(Android 网络接口)。Android SDK 中提供的与网络有关的包如表 10-1 所示。

表 10-1 Android SDK 中与网络有关的包

包	功能描述
java.net.*	提供与网络通信相关的类,包括流和数据包 socket、Internet 协议和常见 HTTP 处理
java.io	虽然没有提供现实网络通信功能,但该包中的类由其他 Java 包中提供的 Socket 和连接使用。它们还用于与本地文件的交互
java.nio	包含表示特定数据类型的缓冲区的类。适用于两个基于 Java 语言的端点之间的通信
org.apache.*	表示许多为 HTTP 通信提供精确控制和功能的包。可以将 Apache 视为开源 Web 服务器
android.net.*	除核心 java.net.* 类以外,包含额外的网络访问 Socket。该包包括 URI 类,后者经常用于 Android 应用程序,而不仅仅是传统的网络操作
android.net.http	包含处理 SSL 证书的类

1. 标准 Java 接口

java.net.* 提供与联网有关的类和接口,包括流和数据包套接字、Internet 协议和常见 HTTP 协议处理。这些类和接口提供了访问 HTTP 服务的基本功能,包括创建 URL

对象和 URLConnection / HttpURLConnection 对象、设置连接参数、连接到服务器、向服务器写入数据以及从服务器读取数据等。其通信可以采用 Post 和 Get 两种方式来实现。URLConnection 对象表示应用程序和 URL 之间的通信连接。程序可以通过它的实例向该 URL 发送请求,读取 URL 引用的资源。

例如,下面的代码片段创建了 URL 对象和 HttpURLConnection 对象,HttpURLConnection 是 URLConnection 的子类。代码中设置了连接参数,连接到服务器并从服务器读取数据。

```
try {
    URL url=new URL("http://www.baidu.com/");            //创建 URL 对象
    HttpURLConnection myconnection=(HttpURLConnection)url.openConnection();
        //创建 URL 连接
    myconnection.setConnectTimeout(10000);               //设置参数
    myconnection.connect();                              //连接服务器
    InputStream is=myconnection.getInputStream();        //取得数据
        …                                                //处理数据
} catch(IOException e) {
    e.printStackTrace();
}
```

2. Apache 接口

HttpClient 是 Apache Jakarta Common 下的子项目,它是一个开源项目,弥补了 java.net.* 灵活性不足的缺点,为客户端的 HTTP 编程提供高效、最新、功能丰富的工具包支持,并且它支持 HTTP 最新的版本和建议。Android 平台引入了 ApacheHttpClient 的同时还提供了对它的一些封装和扩展,例如,设置默认的 HTTP 超时和缓存大小等。Android 平台用的版本是 HttpClient 4.0。对于 HttpClient 类,可以使用 HttpPost 和 HttpGet 类以及 HttpResponse 来进行网络连接。

使用这部分接口的操作方法与 java.net.* 基本类似,主要包括创建 HttpClient、GetMethod / PostMethod 以及 HttpResponse 等对象、设置连接参数、执行 HTTP 操作、处理服务器返回结果等。

下面的代码片段采用 POST 方式实现网络连接:

```
HttpPost httpRequest=new HttpPost(uriAPI);              //创建 HttpRequest 实例
try {
        //发出 HTTP 请求
    httpRequest.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
        //取得 HTTP 响应
    HttpResponse httpResponse=new DefaultHttpClient().execute(httpRequest);
    if(httpResponse.getStatusLine().getStatusCode()==200) {
        //取出应答字符串,若状态码=200,通信正常
      String strResult=EntityUtils.toString(httpResponse.getEntity());
```

```
            myTextView1.setText(strResult);
        }
        else {
            myTextView1.setText("Error:"+httpResponse.getStatusLine().toString());
        }
    } catch(ClientProtocolException e) {
        myTextView1.setText(e.getMessage().toString());
        e.printStackTrace();
    }
```

3. Android 网络接口

android.net.* 包实际上是通过对 Apache 中 HttpClient 的封装来实现的一个 HTTP 编程接口，同时还提供了 HTTP 请求队列管理以及 HTTP 连接池管理，以提高并发请求情况下的处理效率，除此之外还有网络状态监视等接口、网络访问的 Socket、常用的 Uri 类以及有关 WiFi 相关的类等。

下面的代码片段是一个通过 AndroidHttpClient 访问服务器的示例：

```
try {
    AndroidHttpClient client=AndroidHttpClient.newInstance("my_agent");
    HttpGet httpGet=new HttpGet("http://www.test.com/");
        //创建 HttpGet 对象,该对象会自动处理 URL 地址的重定向
    HttpResponse response=client.execute(httpGet);
    if(response.getStatusLine().getStatusCode()==HttpStatus.SC_OK) {
        …                                        //处理数据
    }
    else{
        …                                        //错误处理
    }
    client.close();                              //关闭连接
} catch(Exception e) {
    …                                            //异常处理
}
```

4. 使用 WebView 控件访问网络

在 Android 中，有两种方式访问网页数据：一种方式是使用移动设备上的浏览器直接访问的网络应用程序，这种情况用户不需要额外安装其他应用，只要有浏览器就行；另一种方式则是在用户的移动设备上安装客户端应用程序(.apk)，并在此客户端程序中嵌入 WebView 控件来显示从服务器端下载下来的网页数据。对于前者来说，主要的工作是根据移动设备客户端的屏幕来调整网页的显示尺寸、比例等；而后者需要单独开发基于 WebView 的 Web 应用程序。WebView 控件的详细使用方法见 10.3 节。

10.2　网络资源的访问

由于需要访问网络，本章的操作都需要在 AndroidManifest. xml 配置文件中注册访问网络的权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

另外需要注意的是，Android 4.0 之后系统强制性地不允许在主线程访问网络，否则会出现 android. os. NetworkOnMainThreadException 异常。通常的解决办法是在 onCreate()方法中的 setContentView()语句之后添加以下语句：

```
if(android.os.Build.VERSION.SDK_INT > 9) {
    StrictMode.ThreadPolicy policy=new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
}
```

10.2.1　使用 URL 访问网络

HTTP 是 Web 联网的基础，也是移动设备联网常用的协议之一。HTTP 是建立在 TCP 之上的一种协议，主要用于 Web 浏览器和 Web 服务器之间的数据交换。HTTP 连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。客户向服务器请求服务时，只需传送请求方法和路径，常用的请求方法有 GET、POST 和 HEAD 等。有关 HTTP 的详细介绍，读者可以查阅 RFC2616 或 www. w3c. org。

URL 类位于 java. net 包下，使用的资源可以是简单的文件或目录，也可以是对更复杂的对象的引用。URL 由协议名、主机、端口和资源路径组成。

【例 10-1】 工程 10_URLExample 演示了如何使用 URL 访问网页源码。

首先在 AndroidManifest. xml 配置文件中注册访问网络的权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

在 MainActivity 类中创建 URL 对象，然后通过它的 openStream()方法获取其输入流，再将其中的数据读入字符串，显示到 TextView 控件中。

MainActivity 类的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if(android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy=new StrictMode.ThreadPolicy.Builder().permitAll().build();
```

```
                StrictMode.setThreadPolicy(policy);
            }
            this.setTitle("URL示例");
            TextView show= (TextView)findViewById(R.id.show);
            try {
                InputStream is= new URL("http://www.baidu.com").openStream();
                                //访问百度的 HTML 文件获取输入流
                BufferedReader br= new BufferedReader(new InputStreamReader(is));
                                //读取数据
                String str= null;                   //str 用于读取一行数据
                StringBuffer sb= new StringBuffer();        //StringBuffer 用于存储所有数据
                while((str= br.readLine())!= null) {
                    sb.append(str);                 //读取数据到字符串中
                }
                show.setText(sb.toString());        //显示到 TextView 控件中
            } catch(MalformedURLException e) {
                e.printStackTrace();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
```

示例程序的运行结果如图 10-1 所示。



图 10-1 通过 URL 对象访问网页数据

10.2.2　使用 HttpURLConnection 访问网络

URL 对象的 openConnection()方法返回一个 URLConnection 对象，该对象表示应用程序和 URL 之间的通信连接，程序可以通过 URLConnection 对象向 URL 发送请求、读取 URL 资源。

HttpURLConnection 是 URLConnection 的子类，两者都位于 java.net 包内。使用 HttpURLConnection 对象可以实现 HTTP 连接，具体的用途包括获取 HTML 源码、获取网络图片、获取 XML、发送 GET 请求或 POST 请求、上传文件等。

URLConnection 与 HttpURLConnection 都是抽象类，无法直接实例化对象。其对象主要通过 URL 的 openConnection()方法获得。通常的操作方式是，先通过 URL 对象的 openConnection()方法获取一个 URLConnection 或 HttpURLConnection 对象，然后调用其 getInputStream()方法打开一个 Internet 数据流，读入数据。

【例 10-2】　工程 10_HttpURLConnectionExample 演示了如何使用 HttpURLConnection 访问网页源码。

MainActivity.java 的主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private TextView show;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if(android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy= new StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }
        this.setTitle("HttpURLConnection 示例");
        show= (TextView)findViewById(R.id.show);
        try {
            URL url= new URL("http://www.baidu.com");    //创建一个 URL 对象
            HttpURLConnection conn= (HttpURLConnection)url.openConnection();
                                                        //获取 HttpURLConnection 对象
            conn.setConnectTimeout(6* 1000);            //设置连接超时
            if(conn.getResponseCode() != 200) {         //对响应码进行判断
                show.setText("请求 url 失败!");
            }
            else{
                InputStream is= conn.getInputStream();  //得到网络返回的输入流
                BufferedReader br= new BufferedReader(new InputStreamReader(is));
                String str= null;
                StringBuffer sb= new StringBuffer();          //StringBuffer 用于存储所有数据
                while((str= br.readLine()) != null) {
```

```
                sb.append(str);
            }
            String result= sb.toString();
            conn.disconnect();                              //对文件流操作结束后要及时关闭
            show.setText(result);
        }
    } catch(MalformedURLException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    }
  }
}
```

在程序中设置了连接超时，这是十分必要的，如果网络状态欠佳，Android 系统在超过默认时间会收回资源中断操作，避免了程序长时间的等待。另外，网络读写操作容易产生一些异常，所以在编写网络应用程序时最好捕捉每一个异常以采取相应措施。

示例程序的运行结果如图 10-2 所示。



图 10-2 通过 HttpURLConnection 对象访问网页数据

在 Android 中对文件流的操作要注意，当文件较大时，最好将文件写到 SDCard 而不是直接写到手机内存上，因为手机内存的空间非常有限。另外，对文件流操作结束后要及时将其关闭。

每一次 HttpURLConnection 连接的状态，可以调用 HttpURLConnection.getResponseCode 方法取得当前网络连接的服务器应答代码，或调用 HttpURLConnection.getResponseMessage 取得返回的信息。常见的代码及其对应的信息如表 10-2 所示。

表 10-2 服务器应答代码

ResponseCode	ResponseMessage	说　　明
200	OK	成功
401	Unauthorized	未授权
500	Internal Server Error	服务器内部错误
404	Not Found	找不到该网页

如果要获取网络图片，从 HttpURLConnection 对象中获取输入流读取数据后，调用 BitmapFactory 的 decodeByteArray(byte[] data, int offset, int length)方法将数据转换为图片对象，就可以在 Activity 中使用了。

如果获取的是 XML 数据，还需要使用 XmlPullPaser 对其解析。限于篇幅原因，在此不再赘述，有兴趣的读者可以查阅相关文献。

使用 HttpURLConnection 对象还可以发送 GET 请求或 POST 请求、上传文件等。例如，下面的代码片段实现了采用 POST 方式发送 XML 数据。XML 格式是通信的标准语言，Android 系统可以通过发送 XML 文件传输数据。发送 POST 请求必须设置允许输出和一系列 Request 参数，最好不要使用缓存。

```
byte[] xmlbyte=xml.toString().getBytes("UTF-8");   //将 XML 文件写入到 byte 数组中
URL url=new URL("http://localhost:8080/test/contanct.do?method=readxml");
                                                   //创建 URL 对象，并指定地址和参数
HttpURLConnection conn=(HttpURLConnection)url.openConnection();
conn.setDoOutput(true);                            //设置允许输出
conn.setUseCaches(false);                          //设置不使用缓存
conn.setRequestMethod("POST");                     //设置以 POST 方式传输
conn.setRequestProperty("Connection", "Keep-Alive"); //维持长连接
conn.setRequestProperty("Charset", "UTF-8");       //设置字符集
conn.setRequestProperty("Content-Length", String.valueOf(xmlbyte.length));
                                                   //设置文件的总长度
conn.setRequestProperty("Content-Type", "text/xml; charset=UTF-8");
                                                   //设置文件类型
OutputStream outStream=conn.getOutputStream();
outStream.write(xmlbyte);                          //以文件流的方式发送 XML 数据
```

10.2.3 使用 Socket 进行网络通信

Socket 是网络通信的一种接口。基于不同的协议，有各种不同的 Socket，如基于 TCP 的 Socket、基于 UDP 的 Socket、基于蓝牙协议的 Socket 等。Android 中使用的是 Java 的 Socket 模型，Socket 类在 java.net 包中。

使用 HttpURLConnection 发送数据时，由于系统内部的缓存机制，如果上传较大文件会导致内存溢出。这时可以使用 Socket 发送 TCP 请求，将上传数据分段发送。另外一个重要的区别是，HTTP 连接使用的是"请求/响应"的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据；而 Socket 在双方建立起连接后就可以直接进行数据的传输。

应用程序可以通过 Socket 向网络发送请求或者应答网络的请求，Socket 由两部分组成，一部分是服务器端的 ServerSocket，这个 Socket 主要用来接收来自网络的请求，它一直监听在某一个端口上。端口号取值的范围是 0～65 535，自定义的应用程序通常使用 1024 以上的端口，以避免和其他应用程序的端口冲突。另一部分是客户端的 ClientSocket，这个 Socket 主要用来向网络发送数据。

1. 服务器端编程

通常服务器端编程的一般步骤如下。

步骤 1：首先要在服务器端建立一个 ServerSocket 类的对象并绑定到一个端口上。ServerSocket 对象用于监听来自客户端的 Socket 连接，如果没有连接，它将一直处

于等待状态。

ServerSocket 类常用的构造方法有 3 个。

(1) ServerSocket(int port)：用指定的端口 port 来创建一个 ServerSocket，port 参数必须是一个有效的端口整数值。使用该构造方法创建的 ServerSocket 没有指定 IP 地址，该 ServerSocket 将会绑定到本机默认的 IP 地址。

(2) ServerSocket(int port, int backlog)：用指定的端口 port 和指定连接队列长度创建一个 ServerSocket。

(3) ServerSocket(int port, int backlog, InetAddress addr)：在机器存在多个 IP 地址的情况下，允许通过 addr 这个参数来指定将 ServerSocket 绑定到哪一个 IP 地址。

另外要注意，由于手机无线上网的 IP 地址通常是由移动运营公司动态分配的，一般不会有自己固定的 IP 地址，因此很少在手机上运行服务器端，服务器端通常运行在有固定 IP 的服务器上。

步骤 2：建立 ServerSocket 对象后，调用 accept()方法进入阻塞监听状态直到连接建立。如果接收到一个客户端 Socket 的连接请求，accept()方法将返回一个与客户端连接 Socket 对应的 Socket 对象，然后创建一个线程给该 Socket 对象运行。否则该方法将一直处于等待状态，线程也被阻塞。通常情况下，服务器不应该只接收一个客户端请求，而应该不断地接收来自客户端的所有请求。

步骤 3：ServerSocket 对象接受连接请求后，双方就可以进行通信。通信完成后，ServerSocket 对象回到监听状态，继续监听客户端的请求。

步骤 4：当 ServerSocket 使用完毕后，使用 ServerSocket 的 close()方法来关闭该 ServerSocket。

2. 客户端编程

客户端编程的一般步骤如下。

步骤 1：创建客户端 Socket，指定服务器端 IP 地址与端口号。

客户端通常使用 Socket 的构造方法来连接到指定服务器，Socket 类常用的构造方法有 2 个。

(1) public Socket(InetAddress address, int port)：用服务器端的 IP 地址对象和端口号建立 Socket。

(2) public Socket(String host, int port)：用服务器端的机器名和端口号建立 Socket。

例如，创建连接到本机服务器、9090 端口的 Socket：

```
Socket socket=new Socket("10.0.2.2", 9090);
```

当创建了客户端 Socket 后，就会连接到指定服务器，让服务器上的 ServerSocket 的 accept()方法执行，于是服务器端和客户端就产生一对互相连接的 Socket。这样，Server 和 Client 就可以使用 Socket 进行通信了。

步骤 2：与服务器端进行通信。

当服务器端和客户端产生了对应的 Socket 之后，就可以通过 Socket 进行通信。Socket 提供两个方法来获取输入流和输出流，一个方法是 getInputStream()，返回该 Socket 对象对应的输入流，让程序通过该输入流从 Socket 中取出数据；另一个方法是 getOutputStream()，返回该 Socket 对象对应的输出流，让程序通过该输出流向 Socket 中输出数据。

步骤 3：当 Socket 使用完毕后，使用 close()方法来关闭该 Socket。

【例 10-3】　工程 10_SocketServer 和 10_SocketExample 演示了如何使用 Socket 进行网络通信。

首先创建 Java 工程项目 10_SocketServer，在其中编写类 MySocketServer，实现服务器端的 Socket，主要代码如下：

```
//package 和 import 语句略
public class MySocketServer {
    public static void main(String[] args) throws IOException {
        System.out.println("服务器启动…");
        ServerSocket ss= new ServerSocket(9090);
                //创建一个 ServerSocket,在 9090 端口监听客户端 Socket 的连接请求
        while(true) {                                    //采用循环不断接收来自客户端的请求
            Socket s= ss.accept();
                //每当接收到客户端 Socket 的请求,服务器端也对应产生一个 Socket
            OutputStream os= s.getOutputStream();
                //获取输出流,发送字符串
            os.write("Hello, Welcome!\n".getBytes("utf- 8"));
            os.close();                                  //关闭输出流
            s.close();
                //关闭 Socket
        }
    }
}
```

运行 MySocketServer，结果如图 10-3 所示。此时服务器端 Socket 处于监听状态，直到接收到一个客户端 Socket 的连接请求。当与客户端创建了 Socket 连接后，服务器端 Socket 就会发送字符串“Hello，Welcome!\n”。



图 10-3　启动服务器端监听

接下来，创建 Android 工程项目 10_SocketExample，编写 MainActivity 类，实现客户

端的 Socket，主要代码如下：

```
//package 和 import 语句略
public class MainActivity extends Activity {
    EditText show;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if(android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy= new StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }
        this.setTitle("利用 Socket 通信示例");
        show= (EditText)findViewById(R.id.show);
        try{
            Socket socket= new Socket("10.0.2.2" , 9090);
                //创建连接到本机的服务器、9090 端口的 Socket
            BufferedReader br= new BufferedReader(new InputStreamReader(socket.getInputStream()));
                //将 Socket 对应的输入流包装成 BufferedReader
            String line=br.readLine();                //BufferedReader 数据转换为字符串
            show.setText("来自服务器的数据:\n"+ line);
            br.close();
            socket.close();
                //关闭 Socket
        }catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

该 Activity 运行后，客户端 Socket 会向服务器发出请求，建立连接后获取服务器端发出的数据，其运行结果如图 10-4 所示。



图 10-4　客户端 Socket 获取的数据

10.3　WebView

Android. webkit. WebView 继承自 Android. widget. AbsoluteLayout 类，用于加载和显示 Web 网页。WebView 控件可以被嵌入到应用程序中，实现一个基于 WebKit 浏览器的功能。

在 Activity 中嵌入 WebView 的一般步骤如下。

步骤 1：在 AndroidManifest. xml 配置文件中注册权限 android. permission. INTERNET。

步骤 2：在布局文件中声明 WebView，然后在 Activity 中获取该 WebView 实例。或



在 Activity 中使用 new 操作符实例化 WebView 组件，然后调用 setContentView()方法定义布局。

步骤 3：调用 getSettings()方法取得一个 WebSettings 对象，设置 WebView 属性。例如：

```
mywebview.getSettings().setJavaScriptEnabled(true);  //能够执行 JavaScript 脚本
mywebview.getSettings().setBuiltInZoomControls(true);    //设置允许放大缩小
```

如果需要在 WebView 中使用 JavaScript，则需要设置 WebView 属性使其能够支持 JavaScript。然后，将 JavaScript 与 Android 客户端代码进行绑定，这样就可以由 JavaScript 调用 Android 代码中的方法。例如，JavaScript 代码想利用 Android 的代码来显示一个 Dialog，而不用 JavaScript 的 alert()方法，这时就需要在 Android 代码和 JavaScript 代码间创建接口，从而可以在 Android 代码中实现显示对话框的方法，然后 JavaScript 调用此方法。绑定的具体方法如下。

(1) 创建 Android 代码和 JavaScript 代码的接口，即创建一个类，类中的方法将被 JavaScript 调用。例如：

```
public class JavaScriptInterface {
    Context mContext;
    JavaScriptInterface(Context c) {
            //初始化 Context,供 makeText 方法中的参数来使用
        mContext=c;
    }
    public void showToast(String toast) {
            //创建一个方法,实现显示对话框的功能,供 JavaScript 中的代码来调用
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

(2) 通过调用 addJavascriptInterface 方法，把前面创建的接口类与运行在 WebView 上的 JavaScript 进行绑定。其中第二个参数是为这个接口对象取的名字，以方便 JavaScript 调用。

```
myWebView.addJavascriptInterface(new JavaScriptInterface(this), "Andr_Toast");
```

(3) 在 HTML 中的 JavaScript 部分调用 showToast()方法。

```
<script type="text/javascript">
    function showAndroidToast(toast) {
        Andr_Toast.showToast(toast);
    }
</script>
<input type="button" value="hello" onClick="showAndroidToast('Hello
Android!')"/>
```

另外，Android 中还提供了一个类 WebChromeClient，专门用来辅助 WebView 处理

JavaScript 的对话框、网站图标、网站 Title 和加载进度等。

步骤 4：调用 WebView 的 loadUrl()方法，设置 WebView 要显示的网页。例如：

```
mywebview.loadUrl("http://192.168.1.100:8080");
mywebview.loadUrl("http://www.baidu.com");
mywebview.loadUrl("file:///android_asset/XX.html");                //本地文件，在 assets 文件夹中
```

步骤 5：如果需要在 WebView 中显示网页，而不是在内置浏览器中浏览，则需要调用 setWebViewClient()方法设置 WebView，并重写 shouldOverrideUrlLoading()方法。WebViewClient 是一个专门辅助 WebView 处理各种通知、请求等事件的类。

【例 10-4】 工程 10_WebViewExample 演示了在 Activity 中嵌入 WebView 的用法。

示例工程的实现过程如下。

步骤 1：创建一个新项目 10_WebViewExample。定义布局文件 activity_main. xml，其内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"   >
    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        />
</LinearLayout>
```

步骤 2：定义 MainActivity. java 文件，重写 onCreate()方法，调用 findViewById()方法获得 WebView 的实例对象。然后调用 getSettings()方法取得一个 WebSettings 对象，将 WebView 的 JavaScript 置成可用。如果加载到 WebView 中的网页使用了 JavaScript，就需要在 Websettings 中开启对 JavaScript 的支持，因为 WebView 中默认的是 JavaScript 未启用。

最后，调用 loadUrl(String)加载一个网页。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.setTitle("WebView 示例");
    mywebview=(WebView)findViewById(R.id.webview);
            //获取 WebView 控件实例
    mywebview.getSettings().setJavaScriptEnabled(true);
            //设置 WebView 属性，使其能够执行 JavaScript 脚本
    mywebview.loadUrl("http://www.baidu.com/");
            //加载需要显示的网页
```

```
}
```

步骤 3：在 MainActivity 中添加一个继承自 WebViewClient 内部类 HelloWebViewClient。其作用是启用 Activity 处理自己的 URL 请求。否则，当单击网页中的一个链接时，默认的 Android 浏览器会处理这个 Intent 来显示一个网页，而不是由 Activity 自己来处理。

```
private class HelloWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}
```

WebView 对象初始化之后，为 WebViewClient 设置一个 HelloWebViewClient 的实例。

```
mywebview.setWebViewClient(new HelloWebViewClient()); //设置 Web 视图
```

步骤 4：重写 Activity 类的 onKeyDown()方法。

用 WebView 显示网页，如果不做任何处理，单击设备的“返回”键，整个浏览器会调用 finish()方法结束自身，而不是回退到上一页面。为了让 WebView 支持回退功能，需要重写 Activity 类的 onKeyDown()方法，在此方法中处理 Back 事件。

所以，当单击了网页中的一个链接后，为了按下设备“返回”键时可以导航到前一个页面，重写 onKeyDown()方法：

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if((keyCode==KeyEvent.KEYCODE_BACK) && mywebview.canGoBack()) {
        mywebview.goBack();                              //返回 WebView 的上一页面
        return true;
    }
    return false;
}
```

onKeyDown(int, KeyEvent)回调方法将会在 Activity 中按键被按下的时候被调用。当按下的键是 BACK 并且 WebView 可以回退，即它有历史记录时，就会调用 goBack()方法在 WebView 历史中回退一步。返回 true 表明这个事件已经被处理了。如果条件不满足，这个事件就会被回送给系统。

步骤 5：在 AndroidManifest.xml 配置文件中注册访问网络的权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
```

另外，可以将用户界面设置成没有标题栏的样式，这样给网页更多的显示空间。具体方法是在 AndroidManifest.xml 文件中使用 NoTitleBar 主题：

```
<activity android:name="edu.hebust.zxm.webviewexample.MainActivity"
```

```
android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar">
```

示例程序的运行结果如图 10-5 所示。



图 10-5 在 Activity 中嵌入 WebView

10.4 WebService

10.4.1 WebService 简介

WebService 是一个用于支持网络间不同机器互操作的软件系统，它是一种自包含、自描述和模块化的应用程序，它可以在网络中被描述、发布和调用，可以将它看作是基于网络的、分布式的模块化组件。

WebService 是一种基于 SOAP 的远程调用标准，可以将不同操作系统平台，不同语言、不同技术整合到一起，提供了不同应用程序平台之间的互操作。它使得基于组件的开发和 Web 相结合的效果达到最佳。

对于客户端来讲，可以将 WebService 简单地理解为远程的某个服务器对外公开了某种服务、某个功能或者方法。客户端可以像调用本地方法一样去调用远程服务器上的方法以获得需要的信息，而并不需要关心远程的那个方法是用什么语言编写的、是基于什么平台的。也就是说 WebService 与平台和语言无关。例如，http://www.webxml.com.cn/对外公开了手机号码归属地查询服务，人们只需要在调用该服务时传入一个手机号码，就能立即获取该号段的归属地信息。

WebSerivce 使用 WSDL 描述服务信息。Web 服务描述语言（Web Services Description Language，WSDL）是一种用来描述 Web 服务的 XML 语言，它描述了 Web 服务的功能、接口、参数和返回值等，便于用户绑定和调用服务。例如，前面提到的手机号

码归属地查询的 WebSerivce，其 WSDL 地址是 http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl，在浏览器上访问它就会看到一个 XML 文档，其中包含 SOAP 的版本号、调用它需要传入的参数、返回值的类型等内容。

通常可以通过发送 XML 调用 WebService，其返回结果一般也是 XML 数据。WebService 没有语言限制，只要可以发送 XML 数据和接收 XML 数据即可。

本节介绍在 Android 中如何调用远程服务器端提供的 WebService。

10.4.2　KSoap2 简介

在 Android SDK 中并没有提供调用 WebService 的库，因此，需要使用第三方的 SDK 来调用 WebService。PC 版本的 WebService 客户端库非常丰富，例如 Axis2、CXF 等，但这些开发包并不适合资源有限的 Android 移动设备客户端。对于适合 Android 移动设备的 WebService 客户端类库，最常用是 KSoap2-Android 包，这是 Android 平台上一个高效、轻型的 SOAP 开发包。它的下载地址是 http://code.google.com/p/ksoap2-android/downloads/，进入页面后，找到 jar 文件（例如 ksoap2-android-assembly-2.5.4-jar-with-dependencies.jar）的下载链接，就可以完成下载。

将下载后的 jar 文件复制到 Eclipse 项目工程的 libs 目录中。如果没有该目录，可以新建一个，或放在其他的目录中。然后在 Eclipse 工程中引用这个 jar 包。具体方法是在 jar 文件上右击，执行 Build Path→Add to Build Path 命令；然后再在工程名上右击，执行 Build Path→Config Build Path 命令，这时将弹出如图 10-6 所示的对话框，勾选 ksoap2 jar 包前面的复选框，单击 OK 按钮，就完成了 ksoap2 jar 包的添加。

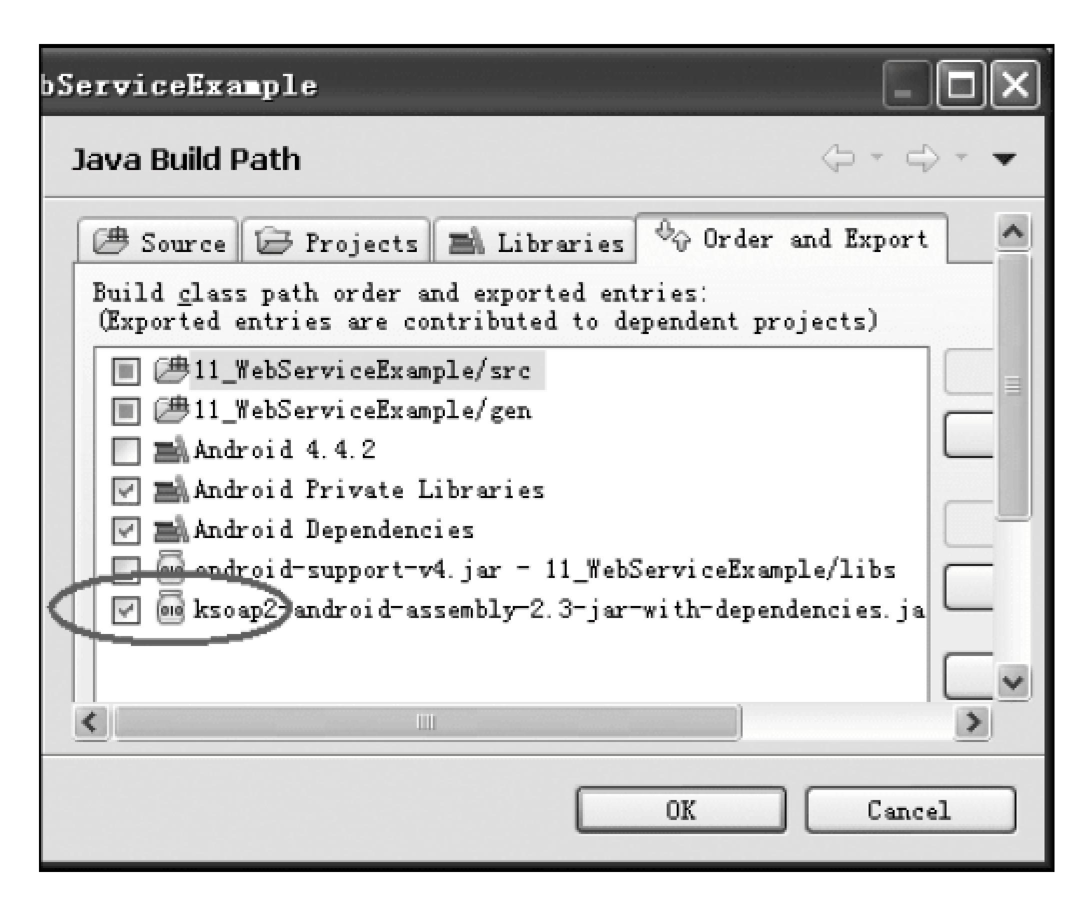


图 10-6　在工程中引用 jar 包

KSoap2 提供如下 4 种接口方式供选择使用。

（1）org.ksoap2.SoapEnvelope：对应于 SOAP 规范中的 SOAP Envelope，其中封装了 head 和 body 对象。

（2）org.ksoap2.serialization.SoapSerializationEnvelope：是对 SoapEnvelope 的一种扩展，对 SOAP 序列化格式规范提供了支持，能够对简单对象自动进行序列化的规范。

（3）org. ksoap2. serialization. SoapObject：构造 SOAP 调用。

（4）org. ksoap2. transport. HttpTransportSE：为使用者屏蔽了 Internet 访问/请求和获取服务器 SOAP 的细节问题。

利用 KSoap2 调用 WebService 的一般步骤如下。

步骤 1：指定 WebService 的命名空间和调用的方法名，格式如下：

```
SoapObject request=new SoapObject("http://service", "getName");
```

SoapObject 类的第 1 个参数表示 WebService 的命名空间，可以从 WebService 的 WSDL 文档中找到其命名空间。第 2 个参数表示要调用的 WebService 方法名。

步骤 2：设置调用方法的参数值，这一步是可选的，如果方法没有参数，可以省略这一步。下面的例子是设置参数值的代码片段：

```
request.addProperty("param1", "value1");
request.addProperty("param2", "value2");
```

addProperty 方法的第 1 个参数表示调用方法的参数名，第 2 个参数是方法的值。

步骤 3：生成调用 WebService 方法的 SOAP 请求信息。该信息由 SoapSerializationEnvelope 对象描述，示例代码如下：

```
SoapSerializationEnvelope envelope=new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.bodyOut=request;
```

创建 SoapSerializationEnvelope 对象时需要通过 SoapSerializationEnvelope 类的构造方法设置 SOAP 的版本号。该版本号需要根据服务端 WebService 的版本号设置。在创建 SoapSerializationEnvelope 对象后，需要设置 SoapSerializationEnvelope 类的 bodyOut 属性，该属性的值就是在第 1 步创建的 SoapObject 对象。

步骤 4：创建 HttpTransportSE 对象。通过 HttpTransportSE 类的构造方法可以指定 WebService 的 WSDL 文档的 URL，例如：

```
HttpTransportSE ht=new HttpTransportSE("http://192.168.0.1:8080/services/SearchProductService");
```

步骤 5：调用 HttpTransportSE 对象的 call()方法实现 WebService 的调用，例如：

```
ht.call(soapAction, envelope);
```

call()方法的第 1 个参数一般为 null，第 2 个参数就是在步骤 3 中创建的 SoapSerializationEnvelope 对象。

步骤 6：使用 getResponse()方法获得 WebService 方法的返回结果，例如：

```
SoapObject soapObject=(SoapObject)envelope.getResponse();
```

10.4.3 在 Android 应用程序中调用 WebService

网站 http://www.webxml.com.cn 上提供了一些免费的 WebService 服务，本节以该网站提供的手机号码归属地查询 WebService 服务为例介绍如何对其进行调用。该



WebService 的方法名为 getMobileCodeInfo，可获得国内手机号码归属地省份、地区和手机卡类型信息。getMobileCodeInfo 方法有两个输入参数：mobileCode=字符串（手机号码，最少前 7 位数字），userID=字符串（商业用户 ID），免费用户为空字符串；返回数据为字符串（手机号码：省份 城市 手机卡类型）。WebService 的调用地址为 http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx？op=getMobileCodeInfo。

【例 10-5】 工程 10_WebServiceExample 演示了如何通过网站提供的 WebService 服务查询手机号码归属地。

示例工程中使用 KSoap2 来调用 WebService，实现过程如下。

步骤 1：查询 WebSerivce 的 WSDL 文档，获取 WebSerivce 的使用方法。本例调用的服务，其 WSDL 地址是 http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx?wsdl，利用浏览器访问这个地址就可以打开对应的 XML 文档，主要内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="http://WebXml.com.cn/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    targetNamespace="http://WebXml.com.cn/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
        <a href="http://www.webxml.com.cn/" mce_href="http://www.webxml.com.cn/" target="_blank"
        >WebXml.com.cn</a>
        <strong>国内手机号码归属地查询 Web 服务</strong>,提供最新的国内手机号码段归属
        地数据,每月更新。<br />
        使用本站 Web 服务请注明或链接本站:
        <a href="http://www.webxml.com.cn/" mce_href="http://www.webxml.com.cn/" target="_blank"
        >
            http://www.webxml.com.cn/
        </a>感谢大家的支持!<br />
    </wsdl:documentation>
    <wsdl:types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://WebXml.com.cn/">
            <s:element name="getMobileCodeInfo">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1" name="mobileCode" type="s:
                        string" />
                        <s:element minOccurs="0" maxOccurs="1" name="userID" type="s:string"
```

```
                        />
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="getMobileCodeInfoResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1" name="getMobileCodeInfoResult"
                        type="s:string" />
                    </s:sequence>
                </s:complexType>
            </s:element>
            ...
        </s:schema>
    </wsdl:types>
    ...
</wsdl:definitions>
```

从上述 WSDL 文档中的标记处可以得到如下信息：该 WebSerivce 所基于的 SOAP 版本是 SOAP 1.2；该 WebSerivce 的命名空间是 http://WebXml.com.cn/；查询手机号码归属地时要调用的方法名称为 getMobileCodeInfo()；调用 getMobileCodeInfo()方法时需要传入两个参数：mobileCode 和 userId；调用 getMobileCodeInfo 方法后，将返回一个名为 getMobileCodeInfoResult 的结果字符串。

步骤 2：新建 Android 工程，引入 ksoap2-android 类库。

步骤 3：设计 MainActivity 的布局文件 activity_main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="请输入手机号(或号码段):" />
    <EditText
        android:id="@+id/mobileNum"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textPhonetic"
        android:singleLine="true"
        android:hint="例如:13933135565(或 1393313)"   />
    <Button
```

```
            android:id="@+id/btnSearch"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="查询"   />
        <TextView
            android:id="@+id/mobileLocation"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"   />
    </LinearLayout>
```

步骤 4：定义 MainActivity。

调用 WebService 之前需要 4 个参数：命名空间、调用的方法名称、EndPoint 和 SOAP Action。在 WebService 的 WSDL 中可以查到命名空间和调用的方法名。EndPoint 通常是将 WSDL 地址末尾的"?wsdl"去除后剩余的部分；而 SOAP Action 通常为命名空间与调用的方法名称的拼接。在调用某些 WebService 时，并不需要传入 SOAP Action，将该参数设置为 null 即可。

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private EditText txtNum;
    private TextView resultView;
    private Button btnSearch;
    private String inputNum;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtNum= (EditText)this.findViewById(R.id.mobileNum);
        resultView= (TextView)this.findViewById(R.id.mobileLocation);
        btnSearch= (Button)this.findViewById(R.id.btnSearch);
        btnSearch.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                inputNum=txtNum.getText().toString().trim();
                if(inputNum.length()<7 || inputNum.length()>11) {
                    //输入数据的有效性检查
                  Toast.makeText(getApplicationContext(), "您输入的手机号码(段)有误,请重新
                  输入!", Toast. LENGTH_LONG).show();
                  txtNum.setText("");
                  resultView.setText("");          //将显示的查询结果清空
                  txtNum.requestFocus();
                  return;
                }
                try {
                    getRemoteInfo(inputNum);                    //调用自定义方法,查询号码信息
                } catch(Exception e) {
```

```
                            Toast.makeText(getApplicationContext(), "查询失败",Toast.LENGTH_LONG).show
                            ();
                        }
                    }
                });
            }
            public void getRemoteInfo(String phoneSec) {
                String nameSpace= "http://WebXml.com.cn/";                  //命名空间
                String methodName= "getMobileCodeInfo";                     //调用的方法名称
                String endPoint= "http://webservice.webxml.com.cn/WebServices/MobileCodeWS.asmx";
                String soapAction= "http://WebXml.com.cn/getMobileCodeInfo";
                SoapObject rpc= new SoapObject(nameSpace, methodName);
                rpc.addProperty("mobileCode", phoneSec);
                rpc.addProperty("userId", "");
                        //设置需调用 WebService 接口需要传入的两个参数
                SoapSerializationEnvelope envelope= new SoapSerializationEnvelope(SoapEnvelope.VER10);
                        //生成调用 WebService 方法的 SOAP 请求信息,并指定 SOAP 的版本
                envelope.bodyOut= rpc;
                envelope.dotNet= true;                  //设置是否调用的是 dotNet 开发的 WebService
                envelope.setOutputSoapObject(rpc);
                HttpTransportSE transport= new HttpTransportSE(endPoint);
                try {
                    transport.call(soapAction, envelope);                   //调用 WebService
                } catch(Exception e) {
                    e.printStackTrace();
                }
                SoapObject object= (SoapObject)envelope.bodyIn;
                        //获取返回的数据
                String result= object.getProperty(0).toString();
                        //提取查询的结果
                resultView.setText(result);                                 //显示 WebService 返回的结果
            }
        }
```

调用 WebService 后,由服务器返回的结果是一个 XML 文件。因为程序中使用了 SoapObject 对象获取返回的数据,所以并不需要通过解析 XML 来获取结果内容。KSoap2 能够将返回的 XML 文档转换成 SoapObject 对象,编程者可以通过操作对象的方式来获取需要的数据。

本例中可以用 object. getProperty("getMobileCodeInfoResult")来取得调用结果。这是因为该 WebService 的 WSDL 文档中明确说明了返回结果是 String 类型,它的名称为 getMobileCodeInfoResult。而有些 WebService 的 WSDL 中并没有返回结果的名称,这时可以调用 SoapObject 对象的 toString()方法来查看返回的内容,例如语句:

```
String result= object.toString();
```

以下语句可以得到的返回结果：

```
getMobileCodeInfoResponse{getMobileCodeInfoResult=1393313:河北 石家庄 河北移动全球通卡;}
```

括号{}里面的内容就是 WebService 的返回结果，这是一个“键-值”对，以等号(=)分隔，通过等号左边的键名 getMobileCodeInfoResult 来获取右边的键值。

在不知道返回结果名称时，有一种更为简便的方法，即直接通过索引下标来获取属性值，例如：

```
String result=object.getProperty(0).toString();
```

有些 WebService 会返回多个值，其获取方法类似，只不过通过 object.getProperty(0)得到的可能仍然是一个 SoapObject，需要不断地调用 getProperty()方法得到全部结果。

步骤 5：由于程序需要访问网络，所以需要在 AndroidManifest.xml 配置文件中注册访问网络的权限。

```
<uses-permission android:name="android.permission.INTERNET"/>
```

示例程序的运行结果如图 10-7 所示。



图 10-7　查询手机号码归属地

10.5　本章小结

本章学习了 Web 应用程序的相关技术和设计方法。利用 URLConnection、HttpURLConnection 或 Socket 可以实现与远程服务器的通信和交互，获取网络中的各种资源；在 Activity 中嵌入 WebView 可以显示从服务器端下载下来的网页数据；利用 WebService 可以实现与服务器上的数据交互。本章学习的重点是网络通信的方法和 WebView、WebService 的用法。

习　　题

1. 使用 HttpURLConnection 从 Internet 上获取一个图片资源，并将其显示在 Activity 中。

2．设计一个利用 Socket 通信的程序，要求建立连接后，ClientSocket 向 ServerSocket 发送字符串“Hello，This is Socket001.”，服务器端接收到这个字符串后，将其打印到控制台。

3．设计一个利用 Socket 通信的程序，要求建立连接后，ClientSocket 向 ServerSocket 发送英文字符串，服务器端接收到这个字符串后，将其转换为大写字母后传回，客户端接收到返回的字符串后将其显示到 Activity 中。

4．编写一个可以发送和接收文本内容的简易聊天程序。

5．WebView 和 WebService 有什么区别？

6．WebView 的 loadData()方法可以实现执行一段 HTML 代码。查阅 Android API 文档，研究 loadData()方法的调用方式，设计一个程序，执行以下 HTML 代码：

```
<html>
    <head></head>
    <body>
        <a  href=http://www.baidu.com>click here</a>
    </body>
</html>
```

7．在示例工程 10_WebViewExample 的基础上，增加一个文本框用于输入网址，增加“前进”、“后退”、“转到”3 个按钮，分别实现网页按照历史记录向前、向后跳转，以及按照文本框中输入的网址直接跳转。

8．网站 http://www.webxml.com.cn 上提供天气预报服务，尝试通过调用 WebService 的方式获取天气预报信息。本书附带的电子文档中提供了该天气预报 WebService 的接口说明。

第11章 综合应用实例

本章介绍几个综合应用的实例，通过学习这些实例，加深读者对基本知识的理解，提高 Android 系统各个功能综合应用的能力。

11.1 简易计算器

【例 11-1】 示例工程 11_CalculatorExample 实现了一个自定义的简易计算器，实现整数和小数的加、减、乘、除运算。

11.1.1 功能分析

工程中使用了 UI 界面控件、菜单、对话框、提示信息等。涉及的知识点包括 XML 布局文件的设计、对按钮单击事件的捕获与响应、菜单的实现等。

本例设计一个简易计算器，实现的计算功能是整数和小数的加、减、乘、除。程序只允许使用界面中提供的按钮，包括 0～9 数字键、小数点和加、减、乘、除运算符输入键、清零和删除键，以及输出结果的“＝”按钮。这些按钮以外的字符全部是非法字符。

按照常规计算器的布局，界面上部是输入和输出区域，下部是功能按钮区域。输入和输出区域不显示光标，没有焦点。文字包括两行，第二行文字较大，实时显示按键生成的计算式。当用户单击“＝”按钮时，显示两行文字：第一行文字较小，为用户生成的计算式；第二行文字较大，显示运算结果。

单击“清零”按钮，显示区域显示 0；单击“删除”按钮，删除最后一次输入的数字或运算符。

11.1.2 设计应用程序的界面布局

为了使程序界面更美观，本例使用 ImageButton 控件实现功能按钮。所以在设计程序之前需要准备 18 个 PNG 图片文件，图片中的内容分别是数字和运算符号。图片文件放置到 drawable 文件夹中。

Activity 的界面布局如图 11-1 所示。界面采用嵌套的 LinearLayout 布局，最外层的 LinearLayout 采用垂直布局，包含 6 个子布局。这 6 个子布局也是 LinearLayout 布局，除第一个以外均采用水平布局。第一个子布局包含一个 TextView 和一个 EditText，用

于显示按键和计算的结果。其余的 5 个 LinearLayout 控制 18 个按钮的布局。为使软件能适应不同分辨率的移动设备，所有控件的 layout_width 和 layout_height 属性都设为 fill_parent，而控制按钮的大小是通过设置 layout_weight 属性值来实现的。这样做的好处是控件的大小只和屏幕大小及控件占屏幕的比例有关。



图 11-1 Activity 的界面

布局文件为 res/layout/activity_main.xml 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="19">
        <TextView android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/editText2"
            android:textSize="20sp"
            android:gravity="top|right"
            android:layout_weight="5"/>
        <EditText android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/editText"
            android:textSize="40sp"
            android:gravity="bottom|right"
            android:layout_weight="2"
            android:text="0"/>
    </LinearLayout>
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="21">
        <ImageButton android:src="@drawable/clean"
            android:id="@+id/buttonclean"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
```

```
            android:layout_weight="1" />
        < ImageButton android:src="@ drawable/delete"
            android:id="@ + id/buttondelete"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
    < /LinearLayout>
    < LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="20">
        < ImageButton android:src="@ drawable/seven"
            android:id="@ + id/button7"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
        < ImageButton android:src="@ drawable/eight"
            android:id="@ + id/button8"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
        < ImageButton android:src="@ drawable/nine"
            android:id="@ + id/button9"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
        < ImageButton android:src="@ drawable/div"
            android:id="@ + id/button_div"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
    < /LinearLayout>
    < !--其余按钮的布局代码与此类似,略-->
< /LinearLayout>
```

11.1.3　设计实现运算的类

在工程中新建一个类文件 Calculate.java,该类的功能是计算用字符串表示的表达式的值。

本例利用堆栈处理用字符串表示的计算式,其基本过程是首先创建两个堆栈,一个用来放数据(numStack),一个用来放运算符(chStack);然后读取运算式,将相应的字符转换为正确的数据格式,压入堆栈。

压栈的过程是从左到右读入算术式,如果读到的是数字,则压入(push)到 numStack

栈中。若读到的是运算符,则先判断 chStack 栈顶元素,若栈顶元素优先级大于读到的算术符,则先将栈顶元素和 numStack 中两个数拿出来计算,再将读到的算术符压入 chStack 中;若读到的算术符优先级大于栈顶元素,则将读到的算术符压入 chStack 中。

如果读到了运算式的最后,则将两堆栈中的内容全拿出来计算,最后结果放在 numStack 中。加号和减号的优先级较低,乘号和除号的优先级较高。因为用到了堆栈,需要在代码之前使用 import 语句引入 java. util. Stack。

```
//package 和 import 语句略
public class Calculate {
    private Stack<Character> chStack;                    //创建一个符号栈
    private Stack<Double> numStack;                      //创建一个数字栈
    private StringBuffer expression;

    public Calculate(String expression) {                //构造方法,初始化计算式
        this.expression=new StringBuffer(expression);    //复制 expression 的内容
        this.chStack=new Stack<Character>();
        this.numStack=new Stack<Double>();
    }

    public double result() throws Exception {            //解析计算式,计算值
        while(this.expression.length()>0) {              //若表达式还没有解析完
            char ch=this.expression.charAt(0);             //获取当前表达式头部的第一个字符
            this.expression.deleteCharAt(0);             //删除取出的那个字符
            double num=0;
            boolean existNum=false;
            while(ch>='0' && ch<='9') {                  //若当前读取到的是数字
                num=num * 10+ch-'0';                       //字符 ASCII 码值转换为实际的数值
                existNum=true;
                if(this.expression.length()>0) {         //继续取数
                    ch=this.expression.charAt(0);
                    this.expression.deleteCharAt(0);
                }
                else {
                    break;
                }
            }
            if(ch=='.')  {
                ch=this.expression.charAt(0);
                this.expression.deleteCharAt(0);
                int i=1;
                while(ch>='0' && ch<='9') {
                    double mi=Math.pow(0.1,i);
                    i++;
                    num=num+(ch-'0') * mi;
```

```
                existNum=true;
                if(this.expression.length()>0) {  //继续取数
                    ch=this.expression.charAt(0);
                    this.expression.deleteCharAt(0);
                }
                else {
                    break;
                }
            }
        }
        if(existNum) {                              //解析完一个数字,则将数字压栈
            this.numStack.push(num);
            //若整个表达式的解析已经结束了,并且以数字结束
            if(this.expression.length()==0 && ch >='0' && ch <='9') {
                break;                              //结束 while 循环
            }
        }
        //若符号栈为空
        if(this.chStack.isEmpty()) {
            this.chStack.push(ch);
            continue;
        }
        switch(ch) {
            case '*':
            case '/':{
            //若符号栈栈顶元素为+、-或者符号栈为空,则意味着符号栈栈顶符号比 ch 优
              先级低,所以,将 ch 压栈。否则,将符号栈栈顶元素弹出来,然后开始计算
                while(this.numStack.size()>=2 && !(this.chStack.isEmpty()|| this.chStack.
                peek()=='+' || this.chStack.peek()=='-')) {
                    this.calcu();
                }
                //若符号栈栈顶元素优先级比 ch 的低
                if(this.chStack.isEmpty()|| this.chStack.peek()=='+' || this.chStack.peek()
                =='-')     {
                    this.chStack.push(ch);
                    continue;
                }
            }
            case '+':
            case '-':{
//若当前符号栈也不为空,则将符号栈栈顶元素弹出来,然后开始计算。因为+、-号的优先级
  最低
                while(this.numStack.size()>=2 &&(this.chStack.peek()=='*'||this.chStack.
                peek()=='/')) {
```

```
                    this.calcu();
                }
                if(this.chStack.isEmpty() || this.chStack.peek()=='('||this.chStack.peek()
                =='+'||this.chStack.peek()=='-') {
                    //若符号栈为空,则将 ch 压栈
                    this.chStack.push(ch);
                    continue;
                }
                else {
                    throw new IllegalArgumentException("表达式格式不合法!");
                }
            }
            default: throw new IllegalArgumentException("运算符非法!");
        }                                           //switch 结束
    }                                               //while 结束
    //若符号栈不为空,则不断地从符号栈和数字栈中弹出元素,进行计算
    while(!this.chStack.isEmpty()) {
        this.calcu();
    }
    //若最终数字栈中仅存一个元素,则证明表达式正确,栈顶元素就是表达式的值
    return this.numStack.size()==1 ?this.numStack.pop(): null;
    }

  private void calcu()throws Exception {        //依据指定的操作数、运算符,进行运算
    double a=this.numStack.pop();               //取出第一个数
    double b=this.numStack.pop();               //取出第二个数
    char op=this.chStack.pop();
    double result=0;
    switch(op) {
        case '+': result=b+a; break;
        case '-': result=b-a; break;
        case '*': result=b * a; break;
        case '/':     {
            if(a==0) {
                throw new ArithmeticException("除数不能为 0!");
            }
            result=b / a;
            break;
        }
    }
    this.numStack.push(result);                 //将运算的结果压栈
  }
}
```

11.1.4　设计 MainActivity 类

MainActivity 类实现计算器程序的主界面，该类继承自 Activity 类，同时实现了 OnClickListener 接口。类中设置了一个字符串 tem，用于暂存输入的计算式。当用户单击“＝”按钮时，将依据这个字符串的内容进行计算。同时它也是计算器的输入输出区域中显示出来的计算式。

首先重写 onCreate()方法，实例化布局中的各控件。接下来对各个按钮绑定监听器，实现算术式的输入功能和计算输出算术式值的功能。清零按钮、回退按钮、等号按钮的功能较特殊，需要单独分别处理。其他的按钮作为基本算式的输入按钮，可看作一类，处理方式类似。

1. “清零”按钮

“清零”按钮的功能是清空输入和输出区域中的内容，其单击事件的主要处理代码如下：

```
public void onClick(View v) {
    edittext.setText("0");
    edittext2.setText("");
    tem= "";
}
```

2. “删除”按钮

“删除”按钮的功能是删除最后一次输入的数字，即当前表达式的最后一个字符，其单击事件的主要处理代码如下：

```
public void onClick(View v) {
    int len= tem.length();
    if(len==0|| len==1) {                    //edittext 中没有任何字符,或只有一个数或运算符
        edittext.setText("0");
    }
    else {
        tem= tem.substring(0,len- 1);        //删除最后一个字符
        edittext.setText(tem);
    }
}
```

3. “＝”按钮

“＝”按钮的功能是计算输入算式的值，并将结果显示在文本框中，同时将算式显示在文本框上方的 TextView 控件中。其单击事件的主要处理代码如下：

```
public void onClick(View arg0) {
```

```
        String str=edittext.getText().toString();          //获得输入的计算式
        Calculate ep=new Calculate(str);                   //计算表达式的值
        try {
            double result=ep.result();
            String result_str=String.valueOf(result);
            edittext2.setText(str+"=    ");
            edittext.setText(result_str);                  //显示结果
            tem=result_str;
        } catch(Exception e) {
            e.printStackTrace();
            edittext.setText("非法输入!");
        }
    }
```

4. 其他按钮

如果单击数字或运算符按钮，则根据按钮的内容在字符串 tem 的末尾增加相应的字符，同时将字符串显示在输出区域，例如当单击 0 按钮时的处理：

```
if(v.equals(button0)) {
    this.firstzero();
    tem=tem+"0";
    edittext.setText(tem);
}
```

其中，firstzero()方法用于处理当数字的第一个字符为 0 时的情况，如果出现这种情况，并且 0 后面不是小数点，这个输入的 0 就不会计入运算式中。方法的定义如下：

```
public void firstzero() {
    if(tem.length()>1) {
        int suml=tem.length()-1;
            //在加、减、乘、除之后是 0,若再输入的是数字则 tem不会增加字符
        if((tem.charAt(suml-1)=='+'||tem.charAt(suml-1)=='-'||
                tem.charAt(suml-1)=='*'||tem.charAt(suml-1)=='/'||)
                &&tem.charAt(suml)=='0') {
            tem=tem.substring(0,suml);
        }
    }
}
```

5. 完善界面的容错功能

为了增强应用程序的可用性，对于算式输入按钮要设置一定的容错功能，以避免生成非法的计算式。本例中设置的容错控制包括：不能连续输入两个小数点，不能连续输入两个运算符，第一个输入的不能是＋、－、×、÷，小数点的输入控制等。当出现上述这些

情况时，输入的内容不会被添加到运算式中。具体代码不再赘述，详见随书源程序。

11.1.5　设计菜单

本例使用 Menu 菜单实现退出、查看版权信息的功能，如图 11-2 所示。

菜单采用 XML 方式实现。先在 res/menu 文件夹中新建 menu.xml 文件，在其中添加菜单项，相关代码如下：



图 11-2　计算器的菜单

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:id="@+id/group1">
        <item android:id="@+id/exit"
            android:title="退出"/>
        <item android:id="@+id/about"
            android:title="关于"/>
    </group>
</menu>
```

重写 MainActivity 中的 onCreateOptionsMenu()方法，在界面中添加菜单。本例中调用 inflate()方法生成菜单，该方法使用一个指定的 XML 资源填充菜单，这里指定的是前一步骤创建的 menu.xml 文件。如果出现错误，该方法会抛出 InflateException 异常信息。相关代码如下：

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater=getMenuInflater();         //获得 menu 容器
    inflater.inflate(R.menu.menu, menu);             //用 menu.xml 填充 menu 容器
    return super.onCreateOptionsMenu(menu);
}
```

重写 onOptionsItemSelected(MenuItem item)方法，实现各个菜单项的功能。单击"退出"选项弹出确认退出对话框，单击"关于"选项显示计算器版权信息对话框。

```
public boolean onOptionsItemSelected(MenuItem item) {
    if(item.getItemId()==R.id.exit) {               //如果单击的是"退出"选项
        Builder exitAlert=new AlertDialog.Builder(MainActivity.this);
        exitAlert.setTitle("警告");
        exitAlert.setMessage("您确定要退出计算机程序吗?");
        exitAlert.setNeutralButton("确定",new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface arg0, int arg1) {
                MainActivity.this.finish();
            }
        });
        exitAlert.setNegativeButton("取消",new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface arg0, int arg1) {
                }
            });
            exitAlert.create();
            exitAlert.show();
        }
        if(item.getItemId()==R.id.about) {             //如果单击的是"关于"选项
            Builder exitAlert=new AlertDialog.Builder(MainActivity.this);
            exitAlert.setTitle("版权声明:");
            exitAlert.setMessage("这是教材的示例程序!\n版本号:1.0");
            exitAlert.setNegativeButton("确定",new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface arg0, int arg1) {
                }
            });
            exitAlert.create();
            exitAlert.show();
        }
        return super.onOptionsItemSelected(item);
    }
```

11.2 音乐播放器

【例 11-2】 示例工程 11_MusicBoxExample 实现了一个功能较完整的音乐播放器。从 MediaStore 获取所有音频文件信息并列表显示,实现音乐文件的播放、暂停、切换等控制。

11.2.1 功能分析

由于本例访问的是媒体库中的数据,所以如果使用模拟器调试运行程序,需要先向模拟器上的 SDCard 中加入音乐文件,然后将模拟器关闭,重新启动以更新媒体库。

工程中使用了 UI 界面控件、菜单、对话框和提示信息等。涉及的知识点包括 XML 布局文件的设计、对按钮单击事件的捕获与响应、内容提供器(媒体库)、音频文件的播放和控制、菜单的实现等。

播放器首先从媒体库中提取音乐文件,然后将每个音乐文件的歌曲名(title)信息显示到 ListVew 中。播放器运行的初始界面如图 11-3(a)所示,单击界面下部的按钮,或单击列表中的歌曲名都可以播放音乐文件。当播放某个音乐文件时,"播放"按钮显示为"暂停"按钮,最上方的文字提示为"正在播放:……",其界面如图 11-3(b)所示。单击"暂停"按钮,音乐暂停播放,同时"暂停"按钮变为"播放"按钮,再次单击该按钮,就会继续播放音乐。

11.2.2 设计应用程序的界面布局

为了使程序界面更美观,本例使用了图片背景和 ImageButton 控件功能按钮。所以





(a)



(b)

图 11-3　播放器的运行界面

在设计程序之前需要准备 7 个 PNG 图片文件，图片中的内容分别是背景图片和按钮图片。图片文件放置到 drawable 文件夹中。

Activity 的界面布局采用嵌套的 LinearLayout 布局，最外层的 LinearLayout 采用垂直布局，包含一个 TextView、ListView 和 LinearLayout 子布局。子布局采用水平布局，包含 5 个按钮。为使软件能适应不同分辨率的移动设备，控件的大小通过设置 layout_weight 属性值来实现。

布局文件为 res/ayout/activity_main.xml 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:background="@drawable/musicback"
    >
   <TextView
      android:id="@+id/txtview"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="歌曲列表:"
      android:textSize="18sp"
      android:layout_weight="20" />
   <ListView
      android:id="@+id/listview"
      android:layout_width="fill_parent"
```

```
            android:layout_height="fill_parent"
            android:layout_weight="4" />
        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="bottom"
            android:layout_weight="1"   >
                <ImageButton
                    android:id="@+id/music_first"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:layout_weight="1"
                    android:src="@drawable/first"   />
                <ImageButton
                    android:id="@+id/music_forward"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:layout_weight="1"
                    android:src="@drawable/foward" />
                <ImageButton
                    android:id="@+id/music_play"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:layout_weight="1"
                    android:src="@drawable/play" />
                <ImageButton
                    android:id="@+id/music_next"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:layout_weight="1"
                    android:src="@drawable/next" />
                <ImageButton
                    android:id="@+id/music_last"
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:layout_weight="1"
                    android:src="@drawable/last"/>
        </LinearLayout>
    </LinearLayout>
```

由于使用了 ListView 控件，所以还需要设置其 Item 的布局文件，布局文件为 res/layout/list_item.xml 文件，内容如下：

```
    <?xml version="1.0" encoding="utf-8"?>
```

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp"
    android:textSize="18sp"
    android:textColor="#0000ff">
</TextView>
```

11.2.3　设计 MainActivity 类

MainActivity 类实现播放器的主界面，该类继承自 Activity 类，同时实现了 OnClickListener 接口。

类中设置了变量 position，用于保存当前播放的媒体文件序号；数组 paths 用于保存歌曲文件存放路径；数组 titles 用于保存歌曲的曲名，如歌曲文件中没有此信息则为文件名。

1. 方法 getMusicFile()

首先设计获取音乐文件的方法 getMusicFile()，该方法的功能是从媒体库中获取所有歌曲的曲名(TITLE)和文件的存放路径(DATA)。

Android 系统通过 ContentProvider 的方式共享媒体库的信息，程序中调用了 android. content. ContextWrapper. getContentResolver ( ) 方法，该方法返回 ContentResolver 对象，然后通过调用该对象的 query()方法获取媒体库的信息。该方法返回 Cursor 对象。

该方法的主要代码如下：

```
private void getMusicFile() {
    cursor=this.getContentResolver().query(
            MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, new String[]
            {MediaStore.Audio.Media.DATA,MediaStore.Audio.Media.TITLE},null,null,null);
                //uri 是系统提供给我们的。查询列:歌曲的保存路径、歌曲名称
    startManagingCursor(cursor);
    if(cursor==null || cursor.getCount()==0) { //判断查询长度是否为空,即查询是否有音乐文件
        txt.setText("没有音乐文件!");
    }
    paths=new String[cursor.getCount()];
    titles=new String[cursor.getCount()];        //数组的长度就是 cursor 的长度
    cursor.moveToFirst();                        //将 cursor 的游标移到第一位
    for(int i=0; i<cursor.getCount(); i++) {     //读取 curor 里面的信息
        paths[i]=cursor.getString(0);            //0 表示索引,即 paths 项
        titles[i]=cursor.getString(1);
        cursor.moveToNext();
    }
```

```
}
```

2. 方法 playMusic()

设计播放音乐文件的方法 playMusic(),该方法的功能是播放从媒体库中获取的第 n 首歌曲。

播放过程通过 MediaPlayer 对象实现。调用该对象的 setDataSource()方法,设置 MediaPlayer 对象播放的数据源,调用其 start()方法,文件开始播放。开始播放后,将界面中的 TextView 控件的显示文字设置为当前播放的歌曲名称,这是从数组 paths 中取得的。另外还要将"播放"按钮的图标改为"暂停"。

该方法的主要代码如下:

```
private void playMusic(int position) {               //播放的方法
    if (mediaPlayer==null) {                          //先要判断 mediaPlayer
        mediaPlayer=new MediaPlayer();                //如果是空的就对它初始化
    }
    try {
        mediaPlayer.setDataSource(paths[position]);   //设置数据源
        mediaPlayer.prepare();                        //设置好数据源之后就可以调用 prepare 这个方法
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    } catch (SecurityException e) {
        e.printStackTrace();
    } catch (IllegalStateException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    mediaPlayer.start();                              //prepare之后就可以调用 start()方法进行播放
    mediaPlayer.setOnCompletionListener(new OnCompletionListener() {
        //设置监听播放下一首的方法,当播放完一首之后会自动播放下一首
            public void onCompletion(MediaPlayer mp) {
                palyNext();
            }
        });
    state=1;
    txt.setText("正在播放:"+titles[position]);
    play.setImageResource(R.drawable.pause);
}
```

3. 方法 palyNext()

设计播放下一首歌曲的方法 palyNext(),该方法的功能是切换到第 $n+1$ 首歌曲。首先判断当前播放的是否是最后一首歌,如果是,则将当前播放位置变量 position 设为



0,循环到第一首歌。否则,将播放位置变量 position 的值加 1,即下一首歌。

该方法的主要代码如下:

```
public void palyNext() {
    stopPlay();
    if(position==titles.length-1) {
                    //判断 position 的值,等于数组的长度,即已经是最后一首歌
        position=0;                     //设为 0,循环到第一首歌
    } else {
        position=position+1;
    }
    playMusic(position);
    play.setImageResource(R.drawable.pause);
}
```

4. 方法 palyForward()

设计播放上一首歌的方法 palyForward(),该方法的功能是切换到第 $n-1$ 首歌曲。首先判断当前播放的是否是第一首歌,如果是,则将当前播放位置变量 position 设为最后一首歌;否则,将播放位置变量 position 的值减 1,即上一首歌。

该方法的主要代码如下:

```
private void palyForward() {
    stopPlay();
    if(position==0) {                   //判断位置是否等于 0
        position=titles.length-1;       //如果等于 0,就循环到最后一首歌
    } else {
        position=position-1;
    }
    playMusic(position);
    play.setImageResource(R.drawable.pause);
}
```

5. 方法 pauseMusic()

设计暂停的方法 pauseMusic(),该方法的功能是暂停当前播放的歌曲,同时将“暂停”按钮的图标设为“播放”。该方法的主要代码如下:

```
private void pauseMusic() {
    if(mediaPlayer !=null) {
        mediaPlayer.pause();
        state=2;
        play.setImageResource(R.drawable.play);
    }
}
```

6. 方法 stopPlay()

设计停止的方法 stopPlay(),该方法的功能是停止播放的歌曲,释放 mediaPlayer 对象的资源。该方法的主要代码如下:

```
private void stopPlay() {
    if(mediaPlayer != null) {
        mediaPlayer.stop();
        mediaPlayer.release();
        mediaPlayer=null;
        state=3;
    }
}
```

7. 重写 onCreate()方法

实例化布局中的各控件,对 ListView 和各个按钮绑定监听器,实现播放器的功能。MainActivity 类的主要代码如下:

```
//package 和 import 语句略
public class MainActivity extends Activity implements OnClickListener{
    public Cursor cursor;                    //使用 Mediastore 进行查询时,需要一个 curor
    private String[] paths;               //保存各音乐文件的路径
    private String[] titles;              ///保存各音乐文件的曲名
    private MediaPlayer mediaPlayer;
    public int position=0;
    private static int state=3;                 //状态变量:1表示播放;2表示暂停;3表示停止
    private ListView music_list;
    private TextView txt;
    private ImageButton first,forward,play,next,last;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.setTitle("我的音乐播放器");
        setupView();
        first.setOnClickListener(this);
        forward.setOnClickListener(this);
        play.setOnClickListener(this);
        next.setOnClickListener(this);
        last.setOnClickListener(this);

        getMusicFile();
        ArrayAdapter<String> adapter=new ArrayAdapter<String> (this, R.layout.list_item, titles);
```

```
        music_list.setAdapter(adapter);
        music_list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view, int mposition, long id) {
                if(mediaPlayer != null) {
                    mediaPlayer.reset();
                }
                position= mposition;
                stopPlay();
                playMusic(position);
            }
        });
    }
    public void onClick(View v) {
        switch(v.getId()) {
        case R.id.music_first:
            stopPlay();
            position= 0;
            playMusic(position);
            break;
        case R.id.music_forward:
            palyForward();
            break;
        case R.id.music_play:
            if(state==1) {
                pauseMusic();
            }else{
                playMusic(position);
            }
            break;
        case R.id.music_next:
                palyNext();
            break;
        case R.id.music_last:
            stopPlay();
            position= titles.length- 1;
            playMusic(position);
            playMusic(2);
            break;
        }
    }
    private void setupView() {
        music_list= (ListView)findViewById(R.id.listview);
        txt= (TextView)findViewById(R.id.txtview);
        first= (ImageButton)findViewById(R.id.music_first);
```

```
            forward= (ImageButton)findViewById(R.id.music_forward);
            play= (ImageButton)findViewById(R.id.music_play);
            next= (ImageButton)findViewById(R.id.music_next);
            last= (ImageButton)findViewById(R.id.music_last);
        }
        private void getMusicFile() {                //获取音乐文件的方法
            …                                        //具体处理略,见本节前述
        }
        private void playMusic(int position) {       //播放的方法
            …                                        //具体处理略,见本节前述
        }
        public void palyNext() {                     //播放下一首歌的方法
            …                                        //具体处理略,见本节前述
        }
        private void palyForward() {                 //播放上一首歌的方法
            …                                        //具体处理略,见本节前述
        }
        private void pauseMusic() {                  //暂停的方法
            …                                        //具体处理略,见本节前述
        }
        private void stopPlay() {                    //停止的方法
            …                                        //具体处理略,见本节前述
        }
        public void finish() {
            super.finish();
            stopPlay();
        }
    }
```

11.2.4 设计菜单

本例使用 Menu 菜单实现退出、查看版权信息的功能,如图 11-4 所示。

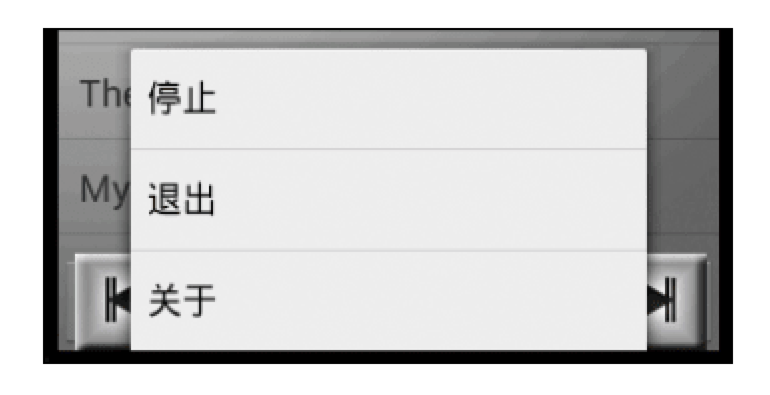


图 11-4 计算器的菜单

菜单采用 XML 方式实现。先在 res/menu 文件夹中新建 menu.xml 文件,在其中添加菜单项,相关代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:id="@+id/group1">
        <item android:id="@+id/menu_exit"
            android:title="停止"/>
        <item android:id="@+id/ menu_exit"
```

```
                android:title="退出"/>
            <item android:id="@+id/menu_about"
                android:title="关于"/>
        </group>
</menu>
```

重写 MainActivity 中的 onCreateOptionsMenu()方法，在界面中添加菜单。本例中调用 inflate()方法生成菜单，该方法指定前一步骤创建的 menu.xml 文件填充菜单，相关代码如下：

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater=getMenuInflater();    //获得 menu 容器
    inflater.inflate(R.menu.menu, menu);         //用 menu.xml 填充 menu 容器
    return super.onCreateOptionsMenu(menu);
}
```

重写 onOptionsItemSelected(MenuItem item)方法，实现各个菜单项的功能。单击“停止”选项音乐播放停止，单击“退出”选项弹出确认退出对话框，单击“关于”选项显示播放器版权信息对话框。具体实现与例 11-1 类似，不再赘述。

11.3　便携日记本

【例 11-3】 示例工程 11_DiaryExample 程序中使用 SQLite 数据库存储日记的日期和内容。每一篇日记由一个唯一的 ID 号标识。

工程中涉及的知识点包括 XML 布局文件的设计，对按钮、ListView 列表项等单击事件的捕获与响应，Activity 之间的跳转和参数传递，SQLite 数据库的操作等。

11.3.1　创建数据库

新建一个类 MyDBOpenHelper，继承自 SQLiteOpenHelper 类。重写其构造方法和 onCreate( )方法。数据库文件存储在/data/data/ edu. hebust. zxm. diaryexample/ databases 目录中，数据库名称为 DB_Diary，其中有用于存储日记信息的数据表 tb_diary。数据表 tb_diary 的结构如表 11-1 所示。

表 11-1 数据表 tb_diary 的结构

列名	数据类型	说　　明
_id	integer	每篇日记的 ID，主键，自动增加
date	text	日记的记录日期
title	text	日记的标题
remark	text	日记的正文

这里要特别注意，因为要使用 SimpleCursorAdapter 适配器将数据表中的数据绑定到 ListView 控件中，所以主键列的列名必须是_ID，这是 SimpleCursorAdapter 特别要求的。

MyDBOpenHelper 类的主要代码如下

```
//package 和 import 语句略
public class MyDBOpenHelper extends SQLiteOpenHelper {
    public MyDBOpenHelper(Context context) {
        //重写构造方法，创建一个名为 DB_Diary 的数据库
        super(context, "DB_Diary", null, 1);
        }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //重写 onCreate()方法，创建数据表，其中 ID 字段作为主键，自动增加
        String sql="create table tb_diary(_id integer primary key autoincrement, date text, title
        text, remark text);";
        db.execSQL(sql);                    //执行 SQL 语句
    }
}
```

实例化这个类，就可以创建相应的数据库了。

11.3.2 界面设计和功能实现

为了实现程序的功能，本例定义了 4 个 Activity 类，分别是 MainActivity、DiaryTextActivity、UpdateDiaryActivity 和 InsertDiaryActivity。4 个 Activity 分别对应主界面、显示日记详细信息、修改日记、新建日记 4 个用户界面。在 MainActivity 中使用 ListView 控件列出了所有日记的 ID、日期和标题，单击列表项则启动 DiaryTextActivity；单击“添加新日记”按钮，则启动 InsertDiaryActivity，添加一条新日记。

在 DiaryTextActivity 中显示选中项的日记详细内容，单击“删除”按钮，可以删除这项日记；单击“修改”按钮，则启动 UpdateDiaryActivity，修改这项日记的标题和内容。

1. MainActivity 类

创建 MainActivity，显示数据库中已有的日记列表。在这个 Activity 中，使用

ListView 控件列出了所有日记的 ID、日期和标题信息，如图 11-5 所示。单击 ListView 中的某条日记，则启动 DiaryTextActivity，显示这条日记的详细内容。单击“添加新日记”按钮，则启动 InsertDiaryActivity，添加一条新日记。

MainActivity 的主要代码如下：

图 11-5　MainActivity 的界面

```
//package 和 import 语句略
public class MainActivity extends Activity {
    private ListView ListDiary;
    private Button btnInsert,btnRefresh,btnClose;
    private SQLiteDatabase dbRead;
    private MyDBOpenHelper dbHelper;
    private Cursor result;

    public void onCreate(Bundle
    savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list);
        this.setTitle("日记列表");
        ListDiary= (ListView)findViewById(R.id.listdiary);
        btnInsert= (Button)findViewById(R.id.btn_insert);
        btnRefresh= (Button)findViewById(R.id.btn_refresh);
        btnClose= (Button)findViewById(R.id.btn_close);
        dbHelper=new MyDBOpenHelper(this);
        dbRead=dbHelper.getReadableDatabase();
                        //获得一个只读的 SQLiteDatabase 对象
        readDiary();
        btnInsert.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent insertIntent=new Intent(MainActivity.this,InsertDiaryActivity.class);
                 startActivity(insertIntent);//启动目标 Activity
            }
        });

        btnRefresh.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                readDiary();
            }
        });

        btnClose.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
```

```
                finish();
            }
        });
}

    protected void readDiary() {
        title_list.clear();
        result=dbRead.rawQuery("select _id, date, title from tb_diary", null);
        int resultCounts=result.getCount();
        if(resultCounts==0 || !result.moveToFirst())
        {
            Toast.makeText(this,"数据库中无数据!", Toast.LENGTH_SHORT).show();
        }
        else
        {
            SimpleCursorAdapter adapter=new SimpleCursorAdapter(getApplicationContext(), R.layout.
            listitem, result,new String[]{"_id","date","title"}, new int[]{R.id.listitem_id,R.id.
            listitem_date,R.id.listitem_title});
            ListDiary.setAdapter(adapter);
                    //将查询到的结果显示到 ListView 控件中
            ListDiary.setOnItemClickListener(new AdapterView.
            OnItemClickListener() {
                public void onItemClick(AdapterView<?>parent, View view, int position, long id) {
                    Cursor cursor=(Cursor)parent.getItemAtPosition(position);
                        //获取单击项的 cursor
                    Intent readIntent=new Intent(MainActivity.this,DiaryTextActivity.class);
                    readIntent.putExtra("selectedID", cursor.getString(0));
                    startActivity(readIntent);
                }
            });
        }
    }
}
```

2. DiaryTextActivity 类

创建显示日记详细内容的 DiaryTextActivity,显示 MainActivity 中被单击的那一项日记的详细内容,运行结果如图 11-6 所示。单击“删除”按钮,可以删除这条日记;单击“修改”按钮,则启动 UpdateDiaryActivity,修改这条日记的标题和内容。





图 11-6　查看日记详细内容

DiaryTextActivity 类的主要代码如下：

```
//package 和 import 语句略
public class DiaryTextActivity extends Activity {
    private TextView ID,date,title,remark;
    private Button Btn_update,Btn_delete,Btn_return;
    private SQLiteDatabase dbRead,dbWrite;;
    private String idStr,dateStr,titleStr,remarkStr,selectedid;
    private MyDBOpenHelper dbHelper;
    private Cursor result;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.diarytext);
        this.setTitle("查看日记详细内容");
        ID= (TextView)findViewById(R.id.Tv_id);
        date= (TextView)findViewById(R.id.Tv_date);
        title= (TextView)findViewById(R.id.Txt_title);
        remark= (TextView)findViewById(R.id.Txt_remark);
        Btn_update= (Button)findViewById(R.id.btn_update);
        Btn_delete= (Button)findViewById(R.id.btn_delete);
        Btn_return= (Button)findViewById(R.id.btn_return);
        selectedid= getIntent().getStringExtra("selectedID");
        dbHelper= new MyDBOpenHelper(this);
        dbRead= dbHelper.getReadableDatabase();
                            //获得一个只读的 SQLiteDatabase 对象
        result= dbRead.rawQuery("select _id, date, title, remark from tb_diary where _id="+
```

```
                selectedid, null);
                result.moveToNext();
                idStr= String.valueOf(result.getInt(result.getColumnIndex("_id")));
                dateStr= result.getString(result.getColumnIndex("date"));
                titleStr= result.getString(result.getColumnIndex("title"));
                remarkStr= result.getString(result.getColumnIndex("remark"));
                ID.setText(idStr);
                date.setText(dateStr);
                title.setText(titleStr);
                remark.setText(remarkStr);

                Btn_update.setOnClickListener(new OnClickListener() {
                    public void onClick(View v) {
                        Intent updateIntent= new Intent(DiaryTextActivity.this,UpdateDiaryActivity.class);
                        updateIntent.putExtra("selectedID", selectedid);
                        startActivity(updateIntent);
                        finish();
                    }
                });

                Btn_delete.setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        dbWrite= dbHelper.getWritableDatabase();
                        String sql= "delete from tb_diary where _id= "+ selectedid;
                        dbWrite.execSQL(sql);
                        Toast.makeText(DiaryTextActivity.this,"日记删除成功!", Toast.LENGTH_SHORT).show
                        ();
                    }
                });

                Btn_return.setOnClickListener(new OnClickListener() {
                    public void onClick(View v) {
                        finish();
                    }
                });
            }
        }
```

3. UpdateDiaryActivity 类

创建修改日记的 UpdateDiaryActivity,界面布局如图 11-7 所示。日记的标题和详细内容分别显示在 EditText 控件中,可以编辑修改其中的文字。单击“确认”按钮,执行 update 语句,修改数据库中的数据。

UpdateDiaryActivity 类的主要代码如下:



图 11-7　UpdateDiaryActivity 的界面布局

```
//package 和 import 语句略
public class UpdateDiaryActivity extends Activity {
    private TextView view_date;
    private EditText txt_title,txt_remark;
    private Button Btn_update,Btn_return;
    private SQLiteDatabase dbRead,dbWrite;;
    private String dateStr,titleStr,remarkStr,selectedid;
    private MyDBOpenHelper dbHelper;
    private Cursor result;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.update);
        this.setTitle("修改日记");
        view_date= (TextView)findViewById(R.id.Tview_date);
        txt_title= (EditText)findViewById(R.id.Txt_title);
        txt_remark= (EditText)findViewById(R.id.Txt_remark);
        Btn_update= (Button)findViewById(R.id.btn_update2);
        Btn_return= (Button)findViewById(R.id.btn_return2);
        selectedid= getIntent().getStringExtra("selectedID");
        dbHelper= new MyDBOpenHelper(this);
        dbRead= dbHelper.getReadableDatabase();
                //获得一个只读的 SQLiteDatabase 对象
        result= dbRead.rawQuery("select _id, date, title,remark from tb_diary where _id="+
       selectedid, null);
        result.moveToNext();
```

```
            dateStr= result.getString(result.getColumnIndex("date"));
            titleStr= result.getString(result.getColumnIndex("title"));
            remarkStr= result.getString(result.getColumnIndex("remark"));
            view_date.setText(dateStr);
            txt_title.setText(titleStr);
            txt_remark.setText(remarkStr);

            Btn_update.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    selectedid= getIntent().getStringExtra("selectedID");
                    dbWrite= dbHelper.getWritableDatabase();
                    String sql= "update tb_diary set title= '"+ txt_title.getText()+ "',remark= '"+ txt_
                    remark.getText()+ "' where _id= "+ selectedid;
                    dbWrite.execSQL(sql);
                    Toast.makeText(UpdateDiaryActivity.this,"日记修改成功!", Toast.LENGTH_SHORT).
                    show();
                }
            });

            Btn_return.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    finish();
                }
            });
        }
    }
```

4. InsertDiaryActivity 类

创建添加新日记的 InsertDiaryActivity，界面布局如图 11-8 所示。分别在两个 EditText 控件中输入日记的标题和内容。单击“添加到日记本”按钮，则使用 insert 语句将所输入内容存储到数据库中。其中 EditText 中的文本分别存储到 title 字段和 remark 字段，当前系统日期存储到 date 字段。

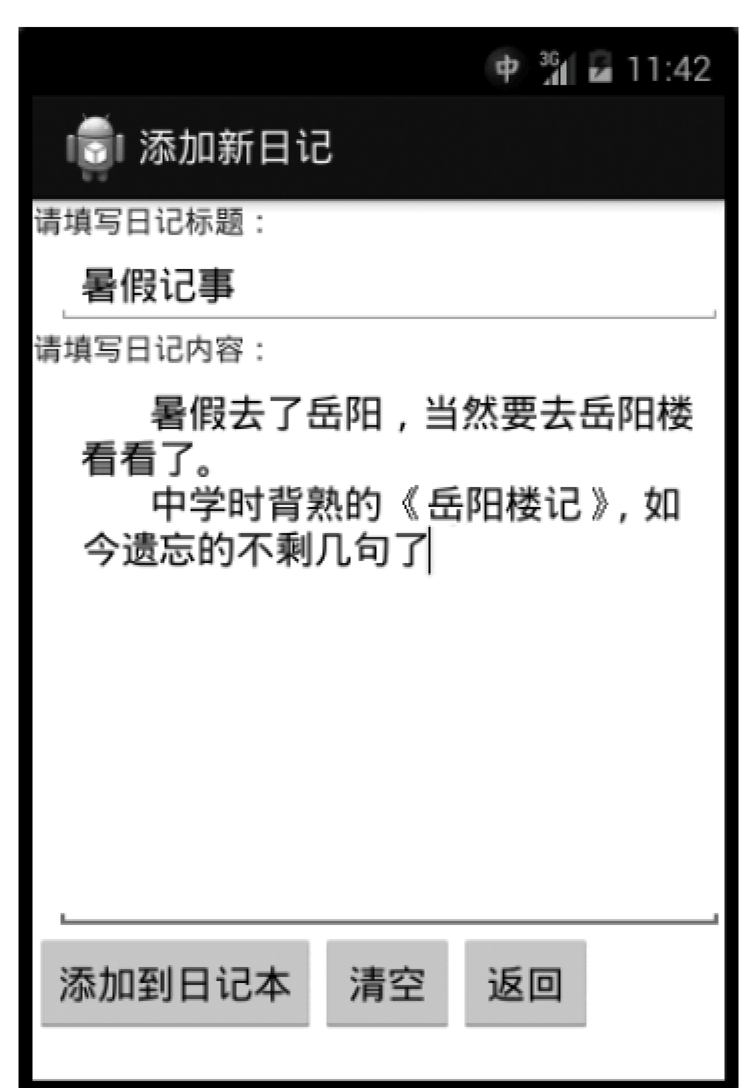


图 11-8 InsertDiaryActivity 的界面布局

InsertDiaryActivity 类的主要代码如下：

```
//package 和 import 语句略
public class InsertDiaryActivity extends Activity {
    private Button BtnInsert,BtnCancel,BtnClear;
    private EditText TxtTitle, TxtRemark;
    private SQLiteDatabase dbWrite;
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.insert);
    this.setTitle("添加新日记");
    TxtTitle= (EditText)findViewById(R.id.Txt_title);
    TxtRemark= (EditText)findViewById(R.id.Txt_remark);
    MyDBOpenHelper dbHelper= new MyDBOpenHelper(this);
    dbWrite= dbHelper.getWritableDatabase();
    BtnInsert= (Button)findViewById(R.id.btn_insert);
    BtnClear= (Button)findViewById(R.id.btn_clear);
    BtnCancel= (Button)findViewById(R.id.btn_cancel);

    BtnInsert.setOnClickListener(new OnClickListener() {
        int year,month,day;
        @Override
        public void onClick(View v) {
            if(TxtTitle.getText().toString().trim().equals("")) {
                Toast.makeText(InsertDiaryActivity.this,"日记标题不能为空,请填写标
                题!", Toast.LENGTH_SHORT).show();
                return;
            }
            Calendar currentTime= Calendar.getInstance();
            day= currentTime.get(Calendar.DAY_OF_MONTH);
            month= currentTime.get(Calendar.MONTH);
            year= currentTime.get(Calendar.YEAR);
        String sql= "insert into tb_diary(date,title,remark)values('"+ String.valueOf(year)+ "
        - "+ String.valueOf(month)+ "- "+ String.valueOf(day)+ "','"+ TxtTitle.getText()+ "','"+
        TxtRemark.getText()+ "');";
            dbWrite.execSQL(sql);
            Toast.makeText(InsertDiaryActivity.this,"新日记添加成功!", Toast.LENGTH_
            SHORT).show();
            TxtRemark.setText("");
            TxtTitle.setText("");
        }
    });
    BtnClear.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            TxtRemark.setText("");
            TxtTitle.setText("");
        }
    });
    BtnCancel.setOnClickListener(new OnClickListener() {
        @Override
```

```
                public void onClick(View v) {
                    finish();
                }
            });
        }
}
```

添加的新日记如图 11-9 所示。



图 11-9 添加的新日记

5. 在 AndroidManifest.xml 中声明所有的 Activity

主要代码如下：

```
<activity android:name= "edu.hebust.zxm.diaryexample.DiaryTextActivity" />
<activity android:name= "edu.hebust.zxm.diaryexample.InsertDiaryActivity" />
<activity android:name= "edu.hebust.zxm.diaryexample.UpdateDiaryActivity" />
```

11.4 本章小结

本章主要介绍了几个 Android 综合应用程序的设计思路和实现方法，这些应用涉及了前几章学习过的界面组件、广播与接收、SQLite 数据库和多媒体应用等。通过这些实例可以加深对基本知识的理解，提高综合应用能力。

习　　题

1. 编写一个备忘录程序，实现备忘信息及提醒时间的输入、删除、修改和保存，以及

预定时间到达后的自动提醒。

2. 编写一个存款管理程序，用户将每笔存款的款额、存入银行的时间和存期，以及支取款额和时间记录在一个数据库中，存期种类和利率如表 11-2 所示。其要求如下：

(1) 每笔存款到期后要给用户一个 AlterDialog 提醒。

(2) 用户随时可以查当前能支取到的存款总额(定期存款随时可支取，但不到期的以活期计算利率)。

(3) 给用户提供参数设置界面，当银行的存款利率发生变化时，能及时设置新的存款利率。利率变化日之前存入的存款按旧利率计算，利率变化日之后存入的存款按新利率计算。

表 11-2　银行存期和利率

存期	年利率/%	存期	年利率/%
活期	0.35	2 年	3.75
3 个月	2.85	3 年	4.25
6 个月	3.05	5 年	4.75
1 年	3.25		

3. 编写一个视频播放器程序，从 MediaStore 获取所有视频信息并列表显示，选择文件后播放视频并实现视频的播放、暂停、停止等控制。

4. 改写示例工程 11_MusicBoxExample，将音乐播放改为用 Service 实现。

参 考 文 献

[1] Google 官方文档. http://developer. android. com/reference/,2014.

[2] 智能手机操作系统. http://baike. baidu. com/,2014.

[3] Java(Java 开发). http://baike. baidu. com/,2014.

[4] (美)H M Deitel,P J Deitel 著. Java 语言程序设计大全[M]. 袁晓靖,等译. 北京:机械工业出版社,1997.

[5] 范春梅,张卫华. XML 基础教程[M]. 北京:人民邮电出版社,2009.

[6] 陈作聪,苏静,王龙. XML 实用教程[M]. 北京:机械工业出版社,2014.

[7] 郭志宏. Android 应用开发详解[M]. 北京:电子工业出版社,2010.

[8] 高凯,王俊社,仇晶. Android 智能手机软件开发教程[M]. 北京:国防工业出版社,2012.

[9] 毋建军,徐振东,林瀚. Android 应用开发案例教程[M]. 北京:清华大学出版社,2013.

[10] 张思民. Android 应用程序设计[M]. 北京:清华大学出版社,2013.

[11] 耿祥义,张跃平. Android 手机程序设计实用教程[M]. 北京:清华大学出版社,2013.

[12] (英)Reto Meier 著. Android4 高级编程[M]. 佘建伟,赵凯译. 北京:清华大学出版社,2013.

[13] 吴亚峰,于复兴,杜化美. Android 应用案例开发大全[M]. 北京:人民邮电出版社,2013.

[14] Datatypes In SQLite Version 3. SQLite 官方文档. http://www. sqlite. org/datatype3. html,2014.

[15] 柳峰的专栏. Android 平台调用 WebService 详解. http://blog. csdn. net/lyq8479/article/details/6428288,2014.

[16] 闵现畅,黄理灿. 基于 Android 平台的 Web 服务技术研究[J]. 工业控制计算机,2011 年第 24 卷第 4 期.

[17] 机锋论坛. http://bbs. gfan. com/.

[18] 安卓网论坛. http://bbs. hiapk. com/forum. php.

[19] Android 开发者社区. http://www. eoeandroid. com/forum. php.

[20] 移动开发者门户. http://www. apkbus. com/portal. php.